ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

APLICACIÓN DE TÉCNICAS DE RECONOCIMIENTO DE PATRONES EN LA SIMULACIÓN DE BATALLAS DE TANQUES COOPERATIVAS

Iñaki Ugarte Gil

José Javier Astrain Escola

Pamplona, 16 de Noviembre de 2011

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

APLICACIÓN DE TÉCNICAS DE RECONOCIMIENTO DE PATRONES EN LA SIMULACIÓN DE BATALLAS DE TANQUES COOPERATIVAS

Iñaki Ugarte Gil

José Javier Astrain Escola

Pamplona, 16 de Noviembre de 2011

1. Introducción

La evolución de las técnicas de inteligencia artificial en general y del reconocimiento de patrones en particular es un tema de extrema importancia en el mundo actual. En sectores que abarcan desde nuevas tecnologías o grandes proyectos innovadores hasta otros que resuelven nuestros problemas de la vida cotidiana, la inteligencia artificial es una respuesta que se desarrolla cada día en mayor medida.

Los últimos avances en inteligencia artificial han permitido el desarrollo de máquinas tan inteligentes que pueden aprobar el Test de Turing; estos nuevos sistemas inteligente nos han dotado de herramientas que nos permiten interactuar mediante sistemas de reconocimiento de voz e incluso de traducción de idiomas (aunque todavía falta mucho para conseguir resultados perfectos). El reconocimiento de patrones en particular forma ya parte de nuestras vidas, ya que se aplica en sistemas de lecturas de todo tipo, desde caracteres, matrículas y señales (Imagen 1.1) hasta imágenes, sonidos y otros parámetros más concretos.

Imagen 1.1: Esquema de funcionamiento del reconocedor de señales de tráfico de un coche

Con la continua evolución de los sistemas informáticos y de procesamiento, es muy normal que se investigue cada vez más en la aplicación de estos procedimientos para la resolución de más problemas de nuestro mundo actual. Uno de los muchos sectores en los que se trabaja con estas técnicas es el de la simulación; la creación de un mundo virtual, de las dimensiones o características adecuadas, en el que se pueda experimentar o comprobar teorías constituye una ayuda o herramienta valiosísima para muchos expertos o investigadores, y por esa razón este sector no ha dejado de desarrollarse.

Además, en algunos casos está muy ligado al entretenimiento público en forma de videojuegos, debido a que sus productos pueden resultar extremadamente parecidos en su función de emular una realidad. Esto ocurre en simuladores que pueden resultar divertidos para el hombre de a pie, como se el caso de la aviación, las competiciones de velocidad e incluso los enfrentamientos militares. Los videojuegos son un sector en el que se mueven muchos millones de euros al año, lo que contribuye al desarrollo de la ciencia si se ponen a la venta algunas versiones de estos simuladores.

Por estas razones resulta muy atractivo trabajar en los proyectos de simulación y videojuegos, y el aplicar técnicas de inteligencia artificial y reconocimiento de patrones resultará en sistemas más parecidos a la naturaleza o al nivel de inteligencia que ésta nos ha concedido, dentro de los límites tecnológicos de que disponemos. La pregunta pertinente en este punto es, ¿en qué campo o sentido realizar una simulación (Imagen 1.2)?

Partiendo de que no se tiene una obligación de ningún tipo ya que el proyecto nace de mi voluntad, se puede escoger el campo de simulación que más convenga al proyecto. Como mi interés en el sector de los videojuegos es grande, me sentía tentado a un ambiente que se pudiera exportar como tal, esto es, que nos diera paso a la creación de un juego interactivo con cierto grado de diversión. Por lo demás no tenía otras preferencias, así que escogí tratar un problema clásico que ha dado pie a muchas simulaciones y videojuegos y que no entraña excesivas dificultades: la simulación de batallas de tanques.

Desde su origen en la Primera Guerra Mundial, los tanques o carros de combate han protagonizado batallas en casi todos los conflictos militares, convirtiéndose en unidades vitales en la Segunda Guerra Mundial, en la que se produjeron batallas masivas de tanques en los que participaban cientos de

vehículos. El problema de los tanques es fácil de tratar en simulaciones en cuanto a lo que recursos gráficos se refiere. No requieren los movimientos constantes y complejos de los seres vivos, si no que su movilidad se limita al avance y retroceso, al giro completo de su torreta y al disparo de proyectiles. Por otro lado tampoco son excesivamente rápidos, por lo que no se requiere de un ambiente de velocidad, y en simulaciones controladas por ordenador, se pueden manejar fácilmente por cualquier persona familiarizada con los controles.

Imagen 1.2: Ejemplo de simulación de una sesión de vuelo con un simulador profesional

Se han creado simuladores y juegos de tanques desde los comienzos de estas técnicas informáticas, de manera que este problema no es nada nuevo. Sin embargo, una de las partes más difíciles de estos programas era el que los vehículos se comportaran como si estuvieran siendo conducidos por personas, esto es, que tuvieran una buena inteligencia artificial. No siempre se disponía de las herramientas software y hardware para dar cobertura a la inteligencia artificial deseada, de modo que recurría a técnicas más sencillas que daban resultados suficientemente buenos para las expectativas del momento. Con los recursos disponibles ahora (más concretamente de los que yo dispongo para este proyecto) tampoco se va a poder conseguir una inteligencia artificial realmente buena, pero aplicaremos las técnicas aprendidas durante la carrera para crear un buen sistema que parezca inteligente a los ojos del usuario.

De esta manera, en este Proyecto Fin de Carrera se pretende crear un entorno de simulación de una batalla de tanques aplicando técnicas de reconocimiento de patrones e inteligencia artificial para lograr un buen realismo en las decisiones y acciones de cada tanque así como de un bando de tanques en general. Evidentemente no se obtendrá el mejor simulador de este tipo ni el más inteligente, ya que no solo haría falta mejor equipo hardware sino que sería un trabajo para un equipo de muchas personas y que conllevaría mucho tiempo de desarrollo. Pero se demostrará que las técnicas aprendidas en la carrera sirven para la resolución de problemas reales (en este caso simulados) y que se pueden aplicar incluso con el trabajo de una sola persona con resultados dignos y respetables, pudiéndose luego exportar a ambientes de trabajo más productivos y poderosos.

Personalmente, la creación y el desarrollo de videojuegos es un tema que siempre ha despertado mucho interés en mí, y con este proyecto quiero aprovechar para familiarizarme mejor con este terreno y enfrentarme a algunas de las dificultades y retos que ofrece a un estudiante. Me gustaría que este trabajo me diera experiencia en este sector y que sea el inicio de un largo camino en la creación y distribución de sistemas de diversión interactivos.

2. Análisis y diseño

Una de las primeras decisiones que se suelen tomar a la hora de diseñar un programa es si la aplicación va a seguir el paradigma estructurado o si se realizará siguiendo las directrices de la orientación a objetos. Esta decisión suele basarse la conveniencia de un paradigma u otro para el resultado final de nuestro proyecto. La siguiente decisión importante suele ser qué lenguaje de programación se va a utilizar en el proyecto y qué herramientas o programas se utilizarán siguiendo ese lenguaje.

Este proyecto, sin embargo, resulta a priori algo diferente, ya que se trata de la aplicación de técnicas de inteligencia artificial a una simulación o videojuego. En cuanto a la inteligencia artificial, aspecto principal del proyecto, se pueden utilizar muchos paradigmas y lenguajes para modelarla y construirla, obteniendo buenos resultados en cualquier caso. Pero la parte de la simulación va a requerir una herramienta especializada en la creación y manipulación de gráficos, esto es, un motor gráfico. Aunque se podría intentar adaptar un motor gráfico al paradigma y lenguaje elegidos, esta combinación puede representar graves problemas en el desarrollo de la simulación, poniendo en grave peligro el proyecto por temas secundarios como es la creación del entorno y las reglas en la que se llevará a cabo la simulación.

Imagen 2.1: Captura de un videojuego realizado con el motor gráfico Blitz3D

Esta fue la razón por la que tomé la decisión de empezar por determinar la herramienta a utilizar antes incluso del lenguaje o paradigma de programación: para evitar el máximo número de problemas que pudieran surgir de la parte gráfica de la aplicación y poder centrarme en la inteligencia artificial. La herramienta escogida para este fin fue el motor gráfico Blitz3D [6].

Blitz3D (Imagen 2.1) es un motor gráfico desarrollado por la compañía Blitzbasic, especializada en la creación de motores gráficos y otras herramientas de diseño de entornos virtuales y videojuegos. Este "engine" es a la vez potente y fácil de usar, e incorpora en su entorno de desarrollo muchas herramientas y comandos de gran utilidad así como tutoriales y ejemplos que muestran su funcionamiento. Está dotado de un compilador que convierte los programas en código de DirectX, con lo que se pueden crear fácilmente ejecutables para Windows (.exe) que se pueden reproducir automáticamente en casi cualquier ordenador actual con Windows. Trae además un manual muy completo y con ejemplos interactivos que facilita en gran medida la adaptación y la programación en su lenguaje propio, que deriva de una sencilla versión de Basic.

Es la herramienta adecuada para crear una simulación de manera fácil y rápida, y sin que esto derive en multitud de problemas que desvíen el proyecto de su objetivo principal: la inteligencia artificial y el reconocimiento de patrones. Es la herramienta que escogí para este proyecto. Como consecuencia, el paradigma del proyecto es el que se utiliza en el sistema de programación de Blitz3D: el paradigma estructurado. Esto desembocará en programas sencillos y funcionales, de manera que no importará que no se vaya a aprovechar la potencia que concede la orientación a objetos. El lenguaje de programación a utilizar es el propio del motor gráfico, que como ya hemos dicho, es una variante de Basic en el que se han incorporado todas las funciones gráficas que proporciona el motor. No es un lenguaje especializado en técnicas de inteligencia artificial, pero contiene los comandos básicos de matemáticas, así que se pueden crear sistemas a partir de estas básicas herramientas.

Por otro lado, a la hora de diseñar e implementar el proyecto opté por un paradigma de desarrollo en espiral (Imagen 2.2), debido a mi inexperiencia en la creación de este tipo de proyectos. Representa una combinación de

videojuego e inteligencia artificial, y como no estaba familiarizado con este tipo de retos, me pareció lo más conveniente abordar el problema poco a poco en vez de tratar de diseñarlo a priori. Por ello realicé varias fases compuestas por análisis y diseño del problema e implementación y pruebas de la solución. Esto provocó un avance más lento del proyecto pero los distintos objetivos se fueron cumpliendo con fiabilidad y sin encontrar problemas demasiado serios o que no se pudieran resolver por un mal análisis y diseño de la aplicación.

Imagen 2.2: Esquema representativo del paradigma de desarrollo en espiral de software

El hecho de que se haya optado por implementar el proyecto siguiendo un paradigma estructurado supuso que el diseño no viniera realizado en forma de diagramas de clases, secuencia e interacción, como es propio en la orientación a objetos, sino que se diseñasen simplemente las estructuras y funciones que se iban a implementar para alcanzar los objetivos de la aplicación. Cabe decir en este punto que la herramienta Blitz3D y su lenguaje de programación no proporcionan comandos para el trabajo con procesos o hilos, por lo que tampoco viene a cuento trazar ningún diagrama de interacción o secuencia que pudiera regular la relación entre distintos procesos de trabajo. Esta limitación restó quizá algo de potencia a la aplicación, pero también simplificó aún más las cosas, de manera que se pudo centrar el

esfuerzo y el trabajo en la inteligencia artificial de los vehículos y en su interacción con el entorno. En cambio, para facilitar la comprensión del diseño de las distintas partes del proyecto, me pareció una buena idea realizar esquemas simples en pseudocódigo que pudieran explicar el trabajo realizado con las funciones y estructuras de manera que cualquiera pudiera entender el trabajo realizado

Cada una de las fases de análisis, diseño, implementación y comprobación tuvo un enfoque particular, unas características propias que la diferencian de alguna manera del resto de fases del proyecto. No me parece adecuado hablar de una fase de análisis y diseño en particular sin haber estudiado primero todo el proceso de la fase anterior, pues muchas de las decisiones que se tomaron en estas fases de análisis y diseño estuvieron condicionadas por las decisiones y problemas resueltos de las fases anteriores. Por esta razón, los procesos de análisis y diseño del proyecto serán incluidos en el apartado de implementación, en el que se entenderán mucho mejor ya que se explicará en orden cronológico todo el proceso de creación que supuso este proyecto.

Sin embargo, para entrever a priori la construcción del proyecto y su resultado final, se muestra a continuación un resumen de las estructuras de datos creadas en el proyecto y un listado de las funciones que se ejecutan en el mismo. Esto nos permite hacernos una idea del aspecto general del proyecto; una representación esquemática en pseudocódigo de las partes más importantes del bucle principal será el broche final que nos permitirá advertir en conjunto el diseño de esta aplicación.

ESTRUCTURAS DE DATOS:

Las estructuras descritas a continuación son el resultado final del diseño y la construcción del proyecto; la sintaxis del lenguaje con respecto a las estructuras es muy fácil de entender: un tipo o estructura se declara entre los comandos “Type *nombre*” y “End Type” y cada campo que se declara dentro de la misma viene precedido de la palabra “Field”. Además, para declarar una variable de tipo real o float se le añade a dicha variable el caracter “#” mientras que para declarar un string o cadena de caracteres se le añade el símbolo “$”. Para crear una tabla, se ponen corchetes al final de la palabra con el número de casillas de la tabla en su interior, por ejemplo, “*meses[12]*”.

La única peculiaridad de las estructuras del lenguaje de Blitz3D es que sus instancias se crean formando una cola; aunque se pueden crear por separado, a una variable de tipo estructura se le pueden ir añadiendo instancias de estructura automáticamente, y se proporcionan comandos para recorrer fácilmente estas construcciones. Esto resulta muy útil a la hora de crear videojuegos, ya que es posible guardar y usar en una variable de tipo estructura multitud de instancias del mismo tipo, por ejemplo, un ejército de naves alienígenas muy parecidas o, en nuestro caso, todos los tanques de la aplicación. Las estructuras creadas, cuatro para este proyecto, serán descritas a continuación.

- **TANQUE:**

La estructura más utilizada y con más campos del proyecto, representa uno de los tanques del combate. Sus variables guardan valores relacionados con su movimiento, inteligencia, construcción gráfica y otros detalles.

```
; Tipo tanque
Type tanque

    ; Variables gráficas del tanque.
    ; Contienen las distintas partes movibles y visibles del tanque
    Field mesh
    Field carro
    Field textura
    Field torreta
    Field canon
    Field mira
    Field bandera
    Field sprite_vida


    Field ruedas[4] ; El tanque se adapta al terreno apoyado sobre
                    ; cuatro esferas en las esquinas del mismo



    ; Parámetros que determinan la velocidad del tanque para el
    ; movimiento, rotación, disparo...
    Field velocidad#
    Field velocidad_rotacion#
    Field velocidad_rotacion_torreta#
    Field cadencia_tiro#
    Field tiempo_tiro


    ; Variables que se utilizan en los sistemas de Inteligencia
    ; Artificial de los vehículos
    Field objetivo
    Field zona_objetivo
    Field enemigo
    Field estrategia


    ; Otros datos del tanque
    Field vida
    Field equipo
    Field numero

End Type
```

- **PROYECTIL:**

Este tipo sirve para representar las balas que disparan los tanques en el desarrollo de la batalla. Se debe actualizar su posición según su velocidad en cada instante o frame del juego, y se debe comprobar si impactan contra algún elemento y actuar en consecuencia.

```
; Proyectiles
Type proyectil

    Field sprite          ; Dibujo de la bala
    Field velocidad#      ; Velocidad del proyectil
    Field recorrido#      ; Distancia recorrida (tiene un máximo)


    ; Número del tanque que lo ha disparado.
    ; Útil para no dispararse a uno mismo
    Field numero


End Type
```

- **EXPLOSIÓN:**

También hay que regular las explosiones que se producen cuando los proyectiles impactan contra algún elemento o explotan porque ya han recorrido su distancia máxima posible. Se controla simplemente la animación del sprite o imagen y la escala o tamaño del mismo. Cuando la animación concluye se desecha la instancia de la estructura.

```
; Explosiones
Type explosion

    Field entidad                 ; "Objeto" que explota
    Field textura_animada         ; Textura o dibujo de la explosión


    ; Parámetros que controlan la animación de la explosión
    Field duracion_animacion
    Field num_frames
    Field frame_actual
    Field tiempo

    ; Parámetros que controlan la escala de la animación
    Field escala#
    Field velocidad_escala#

End Type
```

- **MALLA_HEXÁGONO:**

El terreno de nuestra aplicación se divide en casillas hexagonales para que los tanques puedan calcular con facilidad los mejores puntos (casillas) a los que dirigirse en todo momento. Cada hexágono proporciona información sobre su situación, desnivel y si tiene algún tanque cercano a sí misma.

```
; Malla de hexagonos
Type malla_hexagono

    Field malla ; imagen gráfica de la casilla

    ; Situación de la casilla
    Field x#
    Field y#
    Field z#

    ; Diferencias de altura con las celdas vecinas
    ; Se usa para calcular el camino a puntos altos pero avanzado
    ; por las cuestas más suaves.
    Field gradiente1#
    Field gradiente2#
    Field gradiente3#
    Field gradiente_max#


    Field numero     ; Número de casilla


    ; Número del tanque más cercano a la casilla.
    ; Útil para que no se choquen los tanques
    Field prohibida

End Type
```

FUNCIONES:

El análisis, diseño e implementación del proyecto siguiendo el paradigma de la construcción en espiral dio como resultado un total de 28 funciones, algunas complejas y de gran importancia y otras muy cortas, sencillas y con poca relevancia. Al igual que en el caso de las estructuras, es muy difícil de explicar cómo y por qué se diseñaron estas funciones de la manera en la que se idearon, así que el análisis y diseño relativos a esta parte también se incluirá en el apartado de implementación.

Por ahora se muestra la lista final de funciones, en la que por el nombre se puede intuir qué es lo que hace o pretende realizar cada procedimiento del programa. La lista se ha dividido en 5 secciones, en las que las funciones comparten el mismo tema o realizan funciones de tipo similar.

a) Funciones de creación de objetos:

Las funciones de creación de objetos son las encargadas de generar instancias de elementos complejos, como los tipos o estructuras. Por ejemplo, para las estructuras de tipo tanque o explosión se necesitan varias líneas de código que pueden (y deben) agruparse en una función para mejorar la eficiencia y cohesión de la aplicación. Además, en el caso de "GenerarHexagonos", se crean de golpe todos los hexágonos correspondientes a las casillas de todo el mapa, con lo que con una sola función se simplifica una gran cantidad de trabajo. Las funciones de creación de objetos de la aplicación son las siguientes:

```
; Funciones de creación de objetos:
Function CrearTanque(x#, y#, z#, equipo)
Function BorrarTanque(tanques.tanque)

Function CrearHexagono(x#,z#)
Function DibujarHexagono(numero,pincel)
Function BorrarHexagono(numero)
Function GenerarHexagonos()
Function BorrarHexagonos()
Function DibujarCeldasPrincipales(num)
```

b) Funciones de actualización:

Estas funciones realizan la actualización constante que se requiere de algunos elementos, como el movimiento de los tanques hacia sus objetivos, el

avance de los disparos o la animación de las explosiones. Se ejecutan cada frame del juego, así que tienen que estar controladas y acotadas y ser muy eficientes en su cometido. Por esa misma razón, tampoco puede haber demasiadas de estas funciones, ya que la aplicación se ralentizaría por exceso de carga de procesamiento.

```
; Funciones de actualización:
Function ActualizarDisparos()
Function ActualizarExplosiones()
Function AplicarGravedadTanques()
Function ActualizarMovimientos()
```

c) Funciones nativas de los tanques:

Los tanques del proyecto, entidades principales de la aplicación, tienen sus propias funciones de control debido a que su manejo o movilidad es bastante compleja, al menos en comparación con el resto de elementos. Funciones tan usuales como ver o protegerse son complejas hasta tal punto que si se aplican cada instante o frame del juego, el rendimiento del mismo se reduce drásticamente. Por lo tanto, estas funciones se ejecutan cada ciertos segundos, de manera que se siguen comportando aceptablemente bien sin que esto afecte al funcionamiento de la aplicación en sí. Las funciones propias de los tanques son las siguientes:

```
; Funciones nativas de los tanques:
Function MoverTanque(tanques.tanque,celda)
Function DispararConCadencia(tanques.tanque)
Function SeleccionarTanque()
Function Protegerse()
Function Ver()
```

d) Funciones de inteligencia artificial:

Las funciones propias de la inteligencia artificial, como su propio nombre indica, se encargan de dotar a los tanques de razonamiento para que sus movimientos y acciones emulen las que realizaría un ser humano si los estuviera manejando. Éstas son los procedimientos más complejos de todo el programa, ya que al fin y al cabo, el proyecto consiste en la aplicación de técnicas de reconocimiento de patrones e inteligencia artificial al entorno virtual generado.

Algunas son tan complejas y pesadas que no se pueden aplicar cada instante del programa sin afectar negativamente al rendimiento, por lo que se ejecutan cada pocos segundos, al igual que algunas de las funciones nativas de los tanques. Sin embargo esta complejidad es necesaria y se amortiza, ya que gracias a estas funciones los tanques se comportan de manera bastante real en muchas de las situaciones a las que se enfrentan.

```
; Funciones de la Inteligencia artificial:
Function CalcularMejorCelda()
Function ValorarHexagonos()
Function AplicarEstrategia()
Function CalcularMaximos()
Function LocalizarTanques()
```

e) Funciones auxiliares:

De muy diverso tipo y contenido, las funciones auxiliares son una ayuda para la ejecución de otras funciones o para tareas muy concretas de la aplicación. Aunque puedan parecer superfluas, resultan muy útiles para la programación del proyecto y para el rendimiento del entorno virtual.

```
; Funciones auxiliares:
Function NumeroCelda(entidad)
Function Disparar(tanques.tanque)
Function Orientar(elemento, elemento_objetivo, velocidad, multX, multY)
Function VerAmigos()
Function VerCambios()
Function Contrario(valor)
```

ESQUEMA:

En el siguiente esquema se explica en pseudocódigo el funcionamiento del programa a nivel muy básico y resumido. El objeto de este esquema es dar a conocer el resultado final de la aplicación en cuanto a los procesos o acciones que se realizan en cada instante del programa. Aunque no se usan las funciones reales antes descritas (es pseudocódigo) se puede intuir fácilmente que se hace en cada parte y relacionarlo con dichas funciones.

El esquema sirve además para hacerse a la idea de que el juego, como casi todos los demás videojuegos del mercado, se ejecuta en un bucle que se repite muchas veces por segundo (en nuestro caso, unas 30) y en el que se calculan todos los movimientos y acciones antes descritos y también se dibuja toda la pantalla con los elementos correspondientes (después de borrar la pantalla anterior). Esto es lo que crea la sensación de videojuego y la posibilidad de atender a los dispositivos de entrada que maneja el usuario (teclado y ratón).

Así, de manera muy básica y esquemática, la aplicación en su parte principal tendría el siguiente aspecto:

```
; Crear e inicializar el juego
CrearTerreno( )
CrearTanques( )

; Bucle principal.
; Se ejecuta unas 30 veces por segundo
Mientras No (Salir) Hacer

        ; Funciones de actualización
        AplicarGravedad( )
        ActulizarDisparosYExplosiones()

        ; Cada segundo
        Si (Segundos = 1) Entonces
                Para Cada (Tanque) Hacer

                        VerYProtegerse( )
                        LocalizarEnemigos( )
                        CalcularMejorCeldaCercana( )
                        Moverse( )

                        Si (Enemigo) Entonces
                                Disparar( )
                        Fin Si
                Fin Para
        Fin Si

        ; Cada 10 segundos
        Si (Segundos = 10) Entonces
                Para Cada (Tanque) Hacer

                        ValorarMapa( )
                        AplicarEstrategia( )
                        CalcularMejorCeldaGlobal( )

                Fin Para
        Fin Si

        ; Resto de funciones gráficas
        TratamientoGrafico( )

Fin Mentras
```

Al igual que en cualquier juego, las funciones de creación y carga de objetos están al principio del programa. Después se entra en un bucle que durará toda la simulación y del que se saldrá cuando el usuario active una secuencia de escape. El código dentro del bucle se ejecuta 30 veces por segundo, dato que constituye los FPS (Frames Per Second) del simulador. Ya dentro del bucle, las funciones de gravedad y actualización de proyectiles y animaciones se ejecutan cada frame o iteración para que el usuario no note estos procesos, que le deben resultar naturales. Las funciones de control y manejo automático de los tanques se ejecutan cada determinadas iteraciones, según queramos que reaccionen los tanques y para controlar que no se produzcan demasiados que el ordenador no pueda soportar.

Este es el funcionamiento básico de la aplicación. Como se puede ver, algunas funciones se ejecutan 30 veces por segundo, otras 1 vez por segundo y otras 1 vez cada 10 segundos. Este planteamiento, que como ya se ha explicado pretende evitar la carga excesiva de procesamiento, es fruto del análisis de una de las primeras fases del juego, pues ya desde el principio se pretendía dividir la carga en el tiempo con este método. Ésta decisión y el resto del análisis y diseño, como ya se ha comentado y razonado, se expondrá en el siguiente apartado: Implementación.

3. Implementación

Tal y como se explicó en la parte de Análisis y Diseño de este documento, se escogió para este proyecto aplicar el paradigma en espiral para ir alcanzando objetivos poco a poco y no pretender conseguir algo para luego descubrir que no se puede alcanzar con los medios disponibles. Esto condujo a la división del proyecto en fases independientes, las cuales se hicieron secuencialmente. En principio desconocía el número de fases en que iba a dividirlo, ya que pretendía ir acercándome a mis objetivos finales en las fases que fuesen necesarias, y al final se obtuvieron 4 fases de trabajo, aunque también hubo casos que se trataron en paralelo y que quedaron fuera del entorno final del programa, por razones que veremos cuando las estudiemos en profundidad.

Como corresponde a un modelo de desarrollo en espiral, cada fase se compuso de análisis del problema o situación, diseño de la solución adecuada, implementación de dicho algoritmo y ejecución de pruebas para la verificación del código. En los siguientes apartados se explica todo el trabajo realizado al respecto en cada una de las fases; cada fase se realizó siguiendo un objetivo concreto, aunque en su creación surgieran otros aspectos de la aplicación. Por lo tanto cada fase tiene un nombre característico que indica qué se perseguía en su elaboración. Las 4 fases del proyecto son: "Fase 1: *Tanque movible sobre el terreno*", "Fase 2: *Múltiples tanques destruibles*", "Fase 3: *inteligencia artificial local*" y "Fase 4: *inteligencia artificial global y mejoras*". Las fases paralelas, al final de este apartado, son "*Algoritmo genético para la optimización del tanque*" y "*Reconocedor de tanques*".

Cabe destacar que en las fases de análisis y diseño creadas en la aplicación no había un cliente real al que fuera destinado el programa final o que invirtiera en el mismo. Quien marcaba las pautas en cuanto al diseño, comportamiento y resultado final del programa era yo mismo, de modo que en aspecto de requisitos, es mi opinión la que se escogió en las decisiones tomadas. El hecho de que sea yo mismo quien impuso los requisitos facilitó el que los entendiera perfectamente y el que se adaptasen a lo que yo era capaz de abordar, diseñar y posteriormente programar. Como además estaba familiarizado con el motor gráfico y su lenguaje, sabía bastante bien lo que podía y no podía hacer y tenía algunas ideas sobré cómo hacerlo, así que los requisitos fueron en todo momento bastante adecuados y acertados para el proyecto, al menos desde mi punto de vista.

Fase 1: Tanque movible sobre el terreno

Para un proyecto de simulación en el que se van a aplicar técnicas de reconocimiento de patrones a los entes movibles e inteligentes, primero hay que generar un entorno virtual en el que dichos entes puedan emular las acciones y funciones que llevarían a cabo en ese aspecto de la vida real. En relación con nuestro proyecto, un simulador de batallas de tanques por equipos, la primera fase se centro en la carga de tanque y terreno, en los movimientos permitidos al vehículo y en la interacción de usuario, tanque y terreno.

Esta iba a ser la primera fase del proyecto: crear un tanque con posibilidad de movimiento sobre un terreno acotado y determinado (Imagen 3.1). Los requisitos impuestos fueron que el tanque debía poder moverse por un terreno con cuestas, levemente montañoso, con las flechas de dirección del teclado mientras que la torreta del tanques, que debía girar de manera independiente al resto del tanque (lo que se llamó carro), había de ser controlada por el eje X del ratón (de izquierda a derecha). El eje Y del ratón (de arriba abajo) debía controlar el cañón del tanque para que subiese o bajase, de manera que pudiera apuntarse con relativa facilidad a cualquier parte del terreno. Había de ser indispensable que el tanque se moviese con naturalidad y precisión sobre un terreno escarpado, ya que iba a ser uno de los aspectos que harían interesante al proyecto. Una cámara seguiría al tanque desde detrás y un poco elevada con respecto al vehículo, pero también se tendría que poder girar la misma alrededor del tanque para poder ver mejor sus desplazamientos por el terreno y sus detalles.

El diseño en esta fase del proyecto no resultó muy extenso: el se generó con un mapa de alturas, de modo que cargando una simple imagen de grises se elevara el terreno en consecuencia. Blitz3d proporciona comandos para controlar este sistema, por lo que no hacía falta idear nada más. Para el tanque se escogió un modelo que Blitz3D incorpora en su entorno de desarrollo y que es libre para los usuarios, en este caso, desarrolladores de videojuegos. Sin embargo hubo que retocar este modelo para que se le pudieran aplicar los movimientos fijados en los requisitos en tiempo real. Una estructura o tipo representaría el tanque, guardando tanto su malla o forma gráfica como el resto de sus atributos. Los movimientos que necesarios fácilmente proporcionados por la herramienta de programación, así que no era necesario realizar más proceso de diseño en este aspecto ni para esta fase en concreto.

Imagen 3.1: Captura de pantalla del resultado de la Fase 1 del proyecto

Ya en la fase de implementación del código, el problema más complicado encontrado y abordado en esta fase fue que el tanque se inclinara y se alinease al terreno cuando este estuviera en desnivel. Se hace definiendo una colisión entre el tanque y el terreno, de modo que cuando el tanque choque con el terreno no lo atraviese sino que lo supere. El problema es que en Blitz3D las colisiones son del tipo "de esfera a algo", pudiendo ser esto "de esfera a esfera", "de esfera a cubo" y "de esfera a polígono". El tanque no era esférico ni mucho menos, y su centro de gravedad era bajo, por lo que su esfera de colisión quedaba muy baja. Esto significaba que para que el tanque se deslizase sobre el terreno debía ir dentro de una esfera de colisión que rozara el suelo. Sin embargo, con esta colisión sencilla el tanque no se inclina sobre el terreno y se desplaza de manera muy irregular, como se ve en la imagen 3.2.

Imagen 3.2: el tanque no es capaz de inclinarse para subir por el terreno

La primera solución que se barajo fue la de mover la esfera de colisión para ajustarla lo mejor posible a la forma del tanque. Blitz3D no tiene una función para esto, pero se podría llevar a cabo creando una esfera "padre" del tanque invisible. Lo malo de esta solución es que el tanque no se inclinaría al subir por las cuestas, aspecto fundamental mencionado en los requisitos. La solución final escogida fue la de crear 4 esferas invisibles, como si fueran las ruedas del tanque (como en un coche), que sean las que se encarguen de las colisiones y que el tanque simplemente las siga y se alinee a ellas. La idea la fue extraída de un ejemplo de vehículo adjuntado en la documentación del motor gráfico, el cual tiene un algoritmo para alinear el tanque con las "ruedas"; la solución final para este problema se basó en este ejemplo para resolver este problema, aunque fueron necesarios algunos ajustes para poder aplicarse al tanque, ya que éste no es un coche y no debe comportarse como tal. Además, fue necesario que el propio tanque también colisionase con el terreno, ya que si no se hundía en las profundidades del terreno; para esto hizo falta ajustar su esfera de colisión hasta conseguir el resultado más adecuado. El sistema de 4 esferas resultó muy bueno, y tal y como se ve en la imagen 3.3, el tanque ya se adaptaba de manera correcta a cualquier terreno escarpado.

Imagen 3.3: El tanque ya se alinea perfectamente al terreno gracias a las 4 esferas de colisión

El otro problema relevante en esta fase de implementación fue cómo separar las distintas partes del tanque para que se moviesen independientemente, por ejemplo, el cuerpo del tanque y la torreta con el cañón, que debían girar independientemente. En el modelo disponible venía el tanque como un objeto único y sólido, de modo que había que idear algo para dividirlo en trozos o partes movibles.

La primera opción valorada para mover la torreta del tanque fue separarla como un objeto 3D totalmente independiente y cargarlo como hijo del tanque, para que siguiera todos sus movimientos (aunque Blitz3D no es una herramienta de orientación a objetos, si que proporciona esta clase de herencia). Tras un estudio y reflexión de esta solución, resultó obvio que habría problemas con las texturas, y de habría que disponer de varios modelos para cada ente 3D de la aplicación. Otra opción era aplicar huesos, cosa que resulta un proceso largo complejo de modelado adicional. La solución implementada fue hallada en la documentación adjunta de la

herramienta Blitz3D. En el entorno de edición 3D utilizado, 3D Studio Max, se pueden disociar ciertas partes de las mallas para crear subobjetos de un ente 3D, siempre que se vinculen con el principal del modelo para que sea su “hijo”. Al cargarlo en Blitz3D, el comando "FindChild" extrae ese subobjeto para utilizarlo como ente independiente, aunque se siguió manteniendo como hijo del ente padre cargado. Fue necesario cargar el tanque como "LoadAnimMesh" (malla animada, aunque no tenía animación alguna) para poder usar "FindChild", pero no se produjeron más inconvenientes. El cañón se separó de la misma forma para poder aplicarle movimientos independientes de igual modo.

Estos fueron los sucesos más importantes que ocurrieron durante la implementación; los resultados obtenidos se correspondieron con los requisitos impuestos, por lo que ya era posible proceder a las pruebas de código. En esta fase del proyecto, solo se podía probar cómo reaccionaba el tanque ante los distintos desniveles del terreno, por lo que las pruebas básicamente consistieron en eso. Después de reajustar las esferas de colisión para mejorar la impresión de los movimientos del tanque se finalizó con éxito esta fase del proyecto. No me extiendo más en el relato de estos aspectos porque creo que la parte de inteligencia artificial es la más interesante e importante del proyecto, aunque se podría explicar aquí todo un proceso de simulación y creación de entornos virtuales.

Fase 2: Múltiples tanques destruibles

Esta fase del proyecto puede no parecer muy importante, ya que una vez se dispone de un tanque en el terreno, parece fácil crear uno o dos más y que se puedan disparar y destruir entre ellos. Sin embargo, este proceso requiere de bastante trabajo ya que se va a tener que implementar un sistema de disparos, proyectiles, colisiones e impactos que debe funcionar con precisión y eficiencia para no perjudicar la experiencia final del usuario. Por lo tanto, es vital realizar trabajo y esfuerzos para completar con éxito esta fase del proyecto.

Imagen 3.4: Captura de pantalla de la aplicación tras acabar la Fase 2 del proyecto

Los requisitos planteados fueron que el tanque pudiese disparar proyectiles con su cañón exactamente hacia el lugar al que estuviera apuntando, que estos recorriesen una distancia concreta a una velocidad determinada y que impactaran sobre el terreno o sobre tanques enemigos (Imagen 3.4). En caso de impactar contra otro tanque, este debería perder algo de vida, y si perdía toda la vida debería ser destruido. En este punto se valoró la opción de implementar el "tiro parabólico", esto es, que los disparos fuesen cayendo por la acción de la gravedad describiendo una trayectoria parabólica.

Aunque la herramienta gráfica permitía que no fuera difícil implementar esta parábola (bastaría con restar algo de altura a cada proyectil en cada frame o instante del juego), era evidente que esto luego traería problemas a la hora de apuntar y disparar con el tanque, tanto para los jugadores como para las máquinas (sobre todo para estas últimas, que todavía se desconocía cómo iban a funcionar exactamente). Por lo tanto se decidió no implementarlo para evitar problemas futuros; al fin y al cabo, el juego podía funcionar perfectamente sin el disparo parabólico y dar una muy buena sensación al usuario final.

Para el diseño ya había ideas surgidas de la fase anterior y de su etapa de implementación: las estructuras funcionaban muy bien. Además, el hecho de que las nuevas instancias se pudieran añadir en una cola en la misma variable hacía muy sencillo el recorrer todas las balas para moverlas levemente hacia delante en cada frame de la aplicación. También sería necesaria una estructura para las explosiones, que se representarían con una animación en 2 dimensiones integrada en un sprite (imagen 2D en un mundo 3D que siempre mira hacia la cámara). Con el sistema de colas también se podría recorrer todas las animaciones para chequear si habían terminado y eliminar las instancias de explosiones finalizadas. Con añadir algunos nuevos atributos a la estructura del tanque, como vida y velocidad de disparo, bastaría para adaptarlos a la nueva versión. Crear nuevos tanques, dado el anterior diseño e implementación del proyecto, iba a ser sencillo y eficiente, por lo que no se realizó diseño adicional en este aspecto.

Así, con las nuevas estructuras ya ideadas y diseñadas, se procedió a la implementación de esta fase del proyecto. El primer obstáculo fue encontrado en el cañón de los tanques. Era el turno de animarlo para que cuando disparase, éste se moviera hacia atrás y adelante, fruto de la explosión que se lleva a cabo al disparar el proyectil. En principio una solución era recurrir a la animación de 3D Max, pues es fácil, rápido y sencillo.

Funcionaba correctamente, si no fuera por le hecho de que si la torreta estaba girada, el cañón se alineaba automáticamente con el tanque, tal y como estaba en el modelo de 3D Max. No podía disparar hacia los lados, lo cual era ridículo e imperdonable en un tanque. Para que funcionase así habría que separar el cañón en un modelo aparte para todos los tanques de la aplicación, aplicarles texturas por separado, y vincularlos en Blitz3D. De modo que la solución final escogida consistió en programar manualmente el movimiento del cañón, que con alguna variable auxiliar era posible y relativamente fácil. El problema fue que conforme iba disparando, el cañón no volvía a su posición inicial, sino que se iba desviando hacia adelante, arriba y abajo, fruto del resto de movimientos del tanque, como se puede apreciar en la Imagen 3.5. Ajustar el movimiento del cañón iba a ser complicado y costoso, y habría que ajustarlo a cada tanque según el modelo, así que se siguió buscando una manera más fácil de conseguirlo.

Imagen 3.5: Tras algunos disparos, el cañón se ha desplazado hacia delante y hacia arriba

La solución encontrada era una respuesta intermedia. Consistía en crear dicho movimiento como animación desde Blitz3D, de manera puramente manual y matemática. Se definieron las claves de la animación como se había hecho en 3D Max pero aplicadas solo al cañón, que ya tenía separado en

Blitz3D (la herramienta proporciona comandos muy útiles y sencillos para realizar este trabajo). Con 4 simples operaciones ya estaba completada la animación deseada, funcionaba perfectamente, y podía tratarse como una animación de Blitz3D, con las ventajas de facilidad de control y manejo que esto ofrece. En la Imagen 3.6, después de infinidad de disparos, se puede ver que con esta animación definida con claves manuales el cañón permanece en su sitio a la espera de la siguiente orden de abrir fuego sin separarse ni un milímetro de la torreta.

Imagen 3.6: Tras muchos disparos el cañón sigue en su sitio y se comporta con normalidad

Para que el usuario jugador pudiera apuntar con el cañón se creó una esfera que se desplaza por el mapa en el punto donde una línea imaginaria que parte del cañón intersecta con el terreno o con algún tanque del mapa. Gracias a Blitz3D se disponía de los comandos para implementar este aspecto sin problemas y sin generar código excesivamente pesado para el programa (la herramienta esta preparada para esto). La parte de los disparos, las colisiones y las animaciones también fue bastante sencilla gracias a las

herramientas del motor gráfico y al análisis y diseño realizado en esta fase. Como no surgieron obstáculos relevantes en estos procesos, no se profundizará más en su implementación. En la imagen 3.7 se puede ver un ejemplo del resultado obtenido.

Imagen 3.7: El tanque disparando. Se puede ver el cañón retraído, un proyectil amarillo, la esfera verde de apuntar y una explosión de proyectil. El programa se ajustó para poder ver a la vez todo esto.

Las pruebas realizadas en esta fase tuvieron que ver con que los disparos fuesen dirigidos hacia donde marcaba la esfera de apuntar, que las animaciones se realizasen correctamente y que los tanques fueran destruidos de manera adecuada. La mayoría de las pruebas se realizaron al mismo tiempo que la implementación, ya que dichos ajustes partían directamente de las líneas de código recién programadas. Se volvió a comprobar que los tanques se movieran correctamente por el terreno y que no surgieran bugs o fallos del juego. No se realizaron pruebas especiales, de modo que se continuó con el proyecto pasando a la parte interesante del mismo: la inteligencia artificial de los tanques.

Fase 3: inteligencia artificial local

Una de las partes más importante de un videojuego de batalla es cómo se comportan los enemigos en la lucha contra el jugador: su nivel de inteligencia artificial. Para la realización de este proyecto, se optó por dividir el proceso de creación de inteligencia en 2 fases, de manera que se pudiera abordar de forma más controlada, simple y efectiva. En esta primera fase se tratará la inteligencia artificial local, esto es, la regulación del comportamiento del tanque con respecto a su entorno más cercano.

Así se llegó a una de las decisiones más importantes de todo el proyecto: elegir el tipo de inteligencia artificial que tendrían los tanques y cómo modelarla. Existen multitud de opciones y técnicas de reconocimiento de patrones [4] que resultan perfectamente aplicables a un simulador de batallas, por lo que la elección no era fácil de ninguna manera.

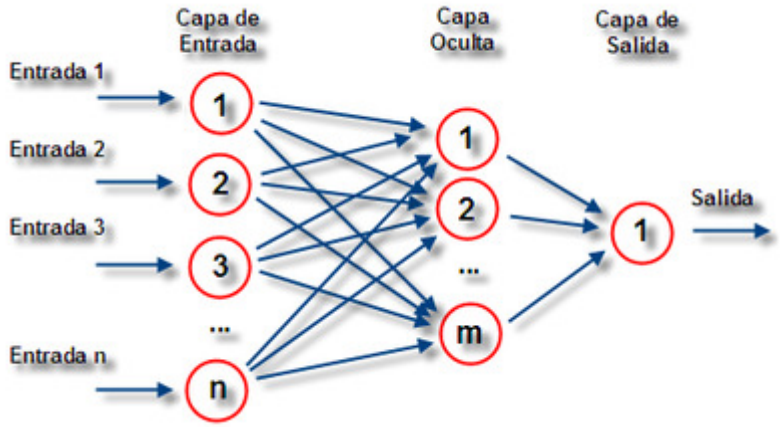


Imagen 3.8: Esquema de representación clásico de una red neuronal

Las redes neuronales [1] (Imagen 3.8) son muy útiles y poderosas para resolver problemas complejos y que no se sabe cómo abordar en principio. Bien diseñadas y programadas son potentes y capaces de enfrentarse a problemas que nunca han estudiado. Su mayor ventaja es que no es necesario que el programador que las usa sepa exactamente cómo funcionan, ya que es la propia red la que determina cuáles son las variables más relevantes y que valores de las mismas permiten identificar un caso u otro y tomar una decisión en consecuencia. Además, se pueden crear y entrenar en una aplicación especializada en su desarrollo y luego exportarlas a casi cualquier lenguaje de programación, como un sistema de instrucciones condicionales

sencillas, aunque después de esta operación no es posible que continúen aprendiendo. Si tras la creación de una red neuronal se comprueba que su resultado no es adecuado, se pueden añadir más casos al entrenamiento para mejorar su índice de acierto, cosa que por lo general suele funcionar. Su mayor desventaja es que si trabajan con bastante variables de entrada y varias capas ocultas, su funcionamiento puede lento, al menos si se quieren incluir en sistemas rápidos o en tiempo real.

Imagen 3.9: Simple esquema representa las bases de la lógica difusa

La lógica difusa [2] (Imagen 3.9) permite más control a la hora de determinar cómo se va a comportar un sistema inteligente. Como gran ventaja, también se encarga el propio sistema de tomar las decisiones, aunque las normas del entorno las haya introducido el desarrollador de dicho sistema. Esto le dota de más velocidad y versatilidad que una red neuronal convencional, aunque como desventaja está el hecho de que hay que definir unas normas de comportamiento que implican cierto estudio del problema previo a su resolución. Además, si el entorno cambia habría que añadir más reglas y seguramente actualizar las existentes para poder adaptarse a los cambios, cosa que convierte el aprendizaje en algo más lento y difícil que en las redes neuronales. También necesita un tiempo de decisión si el sistema valora como entradas muchas variables o aspectos del entorno, aunque sus respuestas suelen ser adecuadas a lo que desean los programadores.

Los algoritmos genéticos [3] (Imagen 3.10) son muy variados y se pueden adaptar a los requerimientos de funcionamiento y velocidad que deseen los desarrolladores. Suelen ser muy fáciles de exportar a cualquier lenguaje de programación, cosa que es siempre una gran ventaja. Su

aprendizaje es constante, con lo que son los que mejor se adaptan a los cambios del entorno, por muy drásticos que sean. Son por estas razones los más versátiles y variados, ofreciendo muchas ventajas y facilidades para su uso. Es sin embargo difícil que alcancen una solución tan buena como los otros métodos mencionados, y aunque la alcanzasen, lo más probable es que evolucionasen hacia otra forma o estado peor, continuamente. Son los más inestables, pudiendo atravesar malas rachas en los que los resultados sean muy malos. Esto no es un problema si se aplican para resolver un problema estático, esto es, que no va cambiando en el tiempo. Pero en entornos en tiempo real puede suponer un problema más serio. Además, suelen precisar de un periodo de adaptación al medio que tampoco suele resultar adecuado en estos entornos de cambio constante.

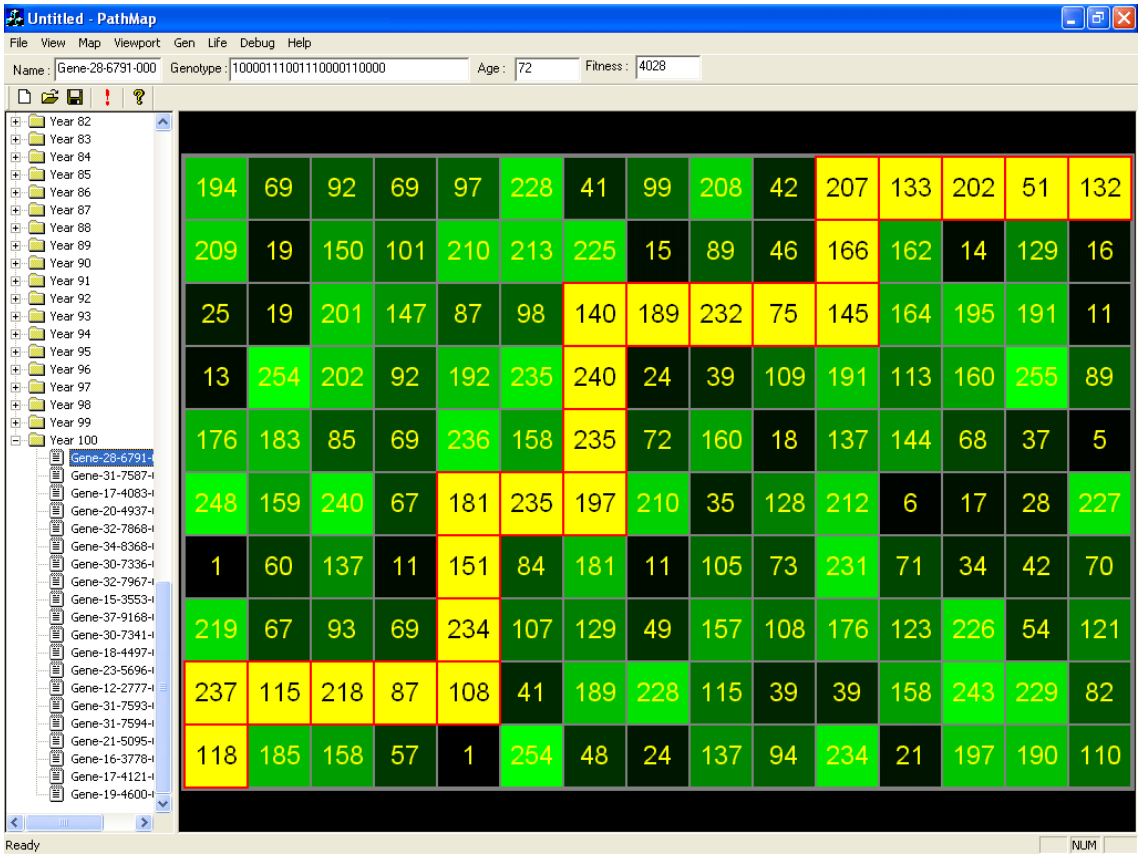


Imagen 3.10: Ejemplo de programa que utiliza un algoritmo genético para calcular un camino

Los sistemas estructurados son los que diseña el desarrollador con su propia solución al problema. Son tan inteligentes como lo fue el programador que los definió. Son la solución más sencilla y eficiente para problemas simples o cotidianos, en los que se pueden imaginar buenas soluciones sin estudiar el problema en profundidad. También son muy adaptables, ya que el propio programador define sus límites y su comportamiento. Además, se adaptan a los requerimientos de eficiencia y complejidad, siempre que a su creador se le ocurra una solución que los cumpla. Son sin embargo malos para problemas complejos en los que requiere mucho estudio averiguar cuáles son las claves que determinarán una buena solución. Incluso después de este estudio, es muy posible que las otras técnicas mencionadas obtengan mejores soluciones, por lo que se habría desperdiciado tiempo y esfuerzo.

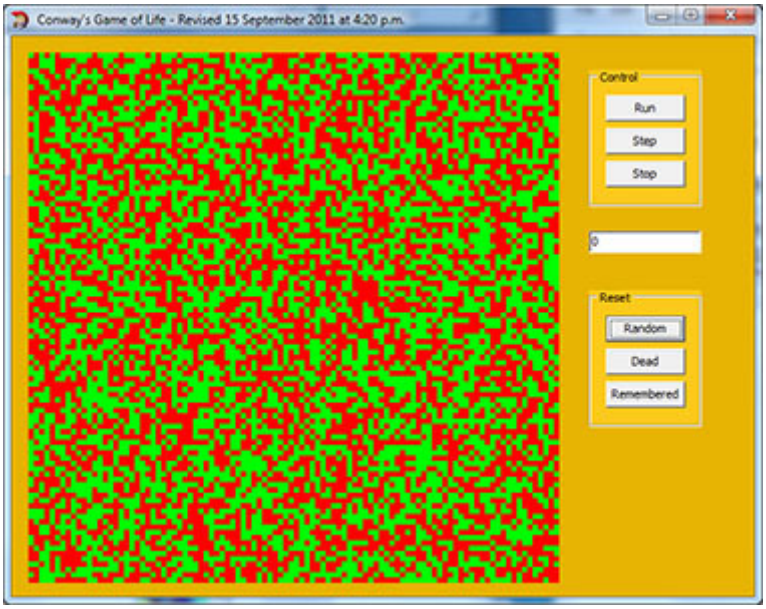


Imagen 3.11: Ejemplo de programa que simula el "Juego de la vida" con un autómata celular

Así que para decidir qué técnica de inteligencia artificial era más conveniente para el proyecto, se estudiaron sus características y requerimientos básicos. Como simulador, se trataba de un programa en tiempo real en el que no era admisible hacer paradas o detener el tiempo. Los ya realizados cálculos de tratamiento de los gráficos y del resto de entidades de la aplicación tampoco permitían un sistema que requiriese mucha carga de procesamiento, y había que adaptarse a los medios de que se disponen en un proyecto fin de carrera, esto es, que tampoco era posible pretender realizar un sistema demasiado grande o excesivamente complejo.

Por estas razones se descartaron las técnicas de redes neuronales, lógica difusa y algoritmos genéticos, como herramientas principales de la inteligencia del juego. Se optó por idear una solución estructurada que se adaptase perfectamente al entorno y a los requerimientos, aunque sus resultados no fueran óptimos. Dentro del razonamiento estructurado, una de las técnicas más atractivas por su sencillez y potencial es el uso de un autómata celular (Imagen 3.11) para calcular en todo momento la mejor situación de cada uno de los tanques. Esto se puede hacer de muchas maneras y teniendo en cuenta multitud de aspectos que se plantearán de aquí en adelante. La primera opción es decidir el alcance del autómata para cada tanque; puede ser que solo interese evaluar como casillas objetivo las casillas más cercanas de cada tanque, ya que al fin y al cabo, para ir a cualquier casilla del mapa va a tener que recorrer las casillas cercanas a su posición actual. Otra opción es decidir sobre cualquier casilla del mapa, sin importarnos tanto por como llegará el tanque a la misma, aunque siempre se pueden puntuar más efectivamente las casillas cercanas para evitar en la medida de lo posible "viajes largos".

La respuesta escogida para el proyecto era una solución combinada; el tanque podría valorar "por zonas" el sitio del mapa al que se quiere dirigir, pero siempre teniendo en cuenta escoger bien entre las casillas cercanas. De esta manera siempre tendría un objetivo bastante claro y bien escogido, y además evaluaría el terreno cercano, tanto de camino al objetivo como una vez llegado a la zona, para escoger las mejores posiciones de la misma.

La manera tradicional de representar casillas en un terreno es mediante una cuadrícula; válida sobre todo para videojuegos en 2 dimensiones, con una buena cuadrícula se podía crear una inteligencia artificial que se adaptase a dicho mapa de una forma muy buena y efectiva. Sin embargo, en la mayoría de entornos 3D, una cuadrícula puede generar movimientos muy rectos o angulosos que dan una sensación muy poco natural al usuario. La opción que se diseñó en esta aplicación consistía en crear una malla de hexágonos para las casillas, algo muy parecido a una típica valla de celdas hexagonales.

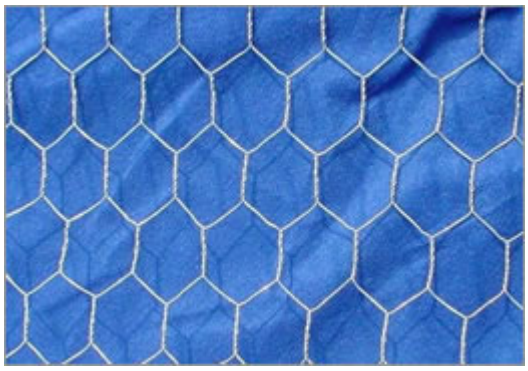


Imagen 3.12: Típica valla hexagonal sobre la que se basa la malla de hexágonos del proyecto

Como se puede ver en la Imagen 3.12, los hexágonos encajan perfectamente unos con otros sin dejar huecos ni espacios, y se pueden adaptar fácilmente a los terrenos montañosos. Aunque la forma de crearlos es más difícil que una sencilla cuadrícula, con este método se consiguen evitar absolutamente los movimientos demasiado rectos o angulosos por el terreno. El algoritmo de generación de hexágonos es simplemente un recorrido por el mapa para ir colocándolos en el lugar y altura adecuados, por lo que no tiene interés alguno analizarlo en profundidad. La parte más complicada era la de escoger una representación que resultara agradable a la vista del usuario y que no fuera muy pesada para no agotar los recursos de la máquina. Lo mejor fue la generación manual de hexágonos mediante vértices y triángulos (utilizada mediante una función para automatizar el proceso). Al disponer estos hexágonos sobre el terreno, se ajustan a la altura que tenga en ese punto para adherirse a las superficies con precisión y eficacia. El resultado obtenido con estas técnicas es el que podemos ver en la Imagen 3.13 y la Imagen 3.14.

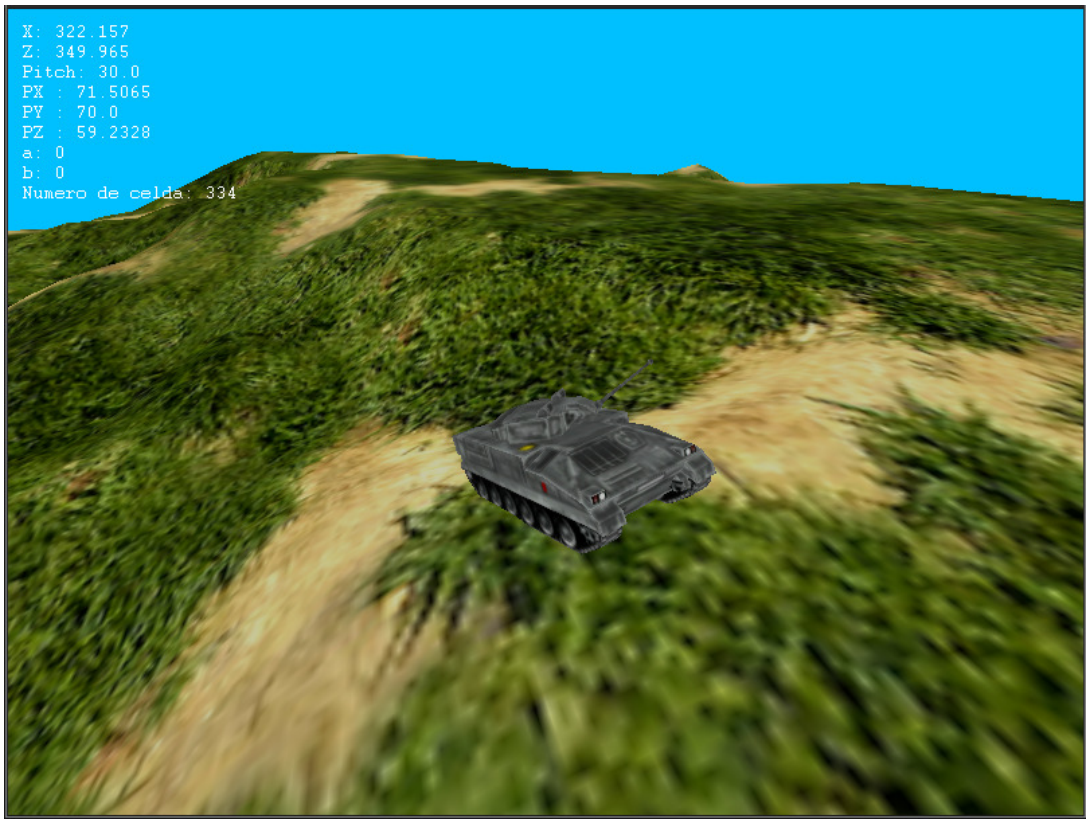


Imagen 3.13: Captura del terreno con su textura y visibilidad normales

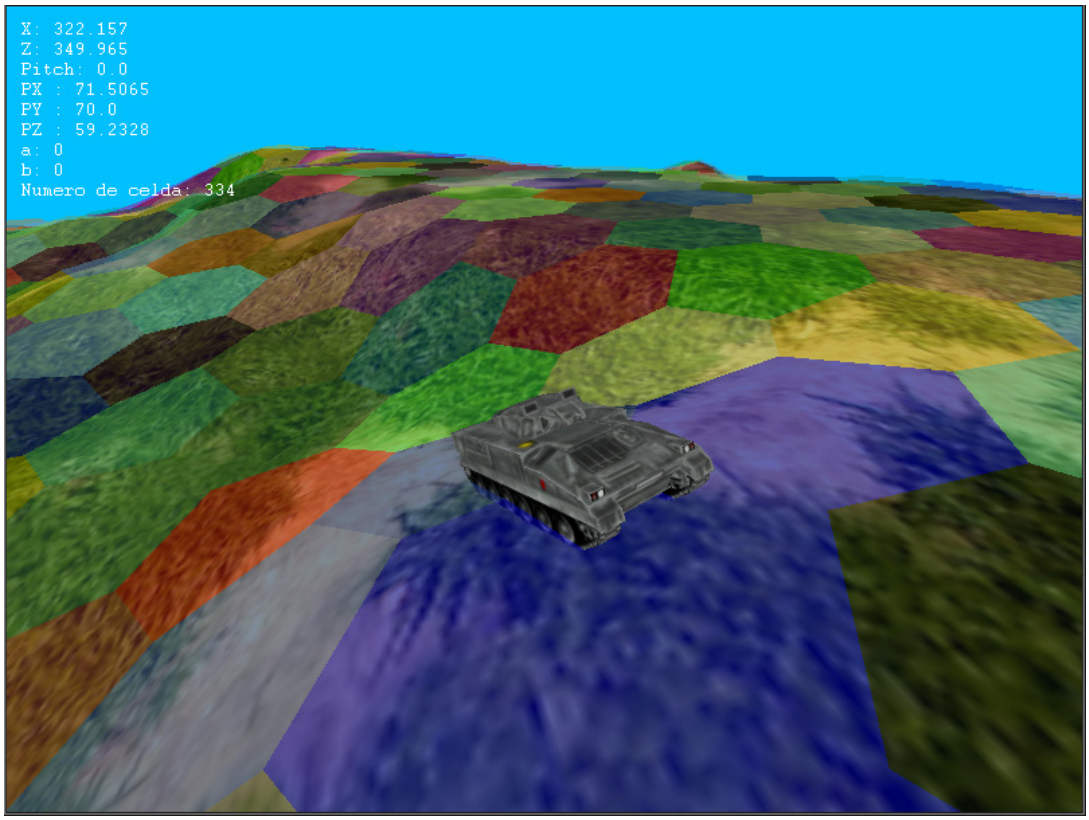


Imagen 3.14: Captura del terreno en el que se ven las casillas hexagonales en que se divide en la aplicación

La malla de hexágonos constituye la base del autómata celular, cuyo comportamiento se definió a continuación. Por un lado, debía tener puntuaciones relacionadas con los desniveles del terreno; hay que tener en cuenta que el mismo terreno debía puntuar de forma distinta para cada tanque, ya que no es lo mismo encontrarse debajo de una montaña que en la cima, y el terreno es sin embargo el mismo. Por eso, como la puntuación del terreno va a depender total y directamente del tanque involucrado, cada tanque deberá tener y calcular su propia puntuación del autómata celular. Con respecto a la representación del terreno, no solo se debía tener en cuenta la altura sino también el desnivel de las casillas de alrededor del tanque, ya que habrá algunos desniveles que el tanque no podrá atravesar, como cuestas demasiado empinadas y, en el peor caso, paredes.

También debía influir en las puntuaciones la "zona objetivo" actual de cada tanque, que ya veremos en el siguiente apartado *inteligencia artificial global y otras mejoras* como se calcula y determina. Se tuvo en cuenta que la mayoría de las veces que se puntúe un terreno será para atravesarlo con el fin de llegar a otra zona del mapa. Por eso, si se está en una zona que no es objetivo, se debían puntuar significativamente las casillas que conduzcan al objetivo especificado. La estrategia concreta del tanque en el momento del cálculo también influiría sobre la puntuación de las casillas. No es lo mismo tratar de esconderse que buscar el punto más alto o perseguir a los enemigos, por lo que se debía tener en cuenta en todo momento.

El último aspecto importante que se tuvo en cuenta sobre el terreno era la situación de aliados y enemigos. Esto vendría determinado tanto por el número de enemigos y aliados en la zona como por la estrategia del tanque en ese instante, ya que las casillas no serán igualmente valiosas si se decide huir del enemigo que si se opta por rodearlo o embestirlo de frente. La selección de las "zonas objetivo" también dependería en gran medida de la estrategia actual que siguiera cada tanque; en esta estrategia también influiría la situación conocida de los tanques aliados y enemigos así como el número de tanques existentes cuya situación se desconozca (en el apartado siguiente se explicarán en profundidad estos aspectos). En resumen, se decidió que para calcular las mejores casillas entre las cercanas al tanque influirían los atributos de altura, gradiente, zona objetivo, estrategia y aliados y enemigos.

Volviendo a la puntuación de cada casilla de una zona, su valor sería un entero que empezaría valiendo cero. Cada aspecto positivo de la casilla sumaría puntos a la misma, mientras que cada aspecto negativo se los restaría. Así tendríamos casillas con valores positivos o negativos, pero el destino será claro: la casilla con más puntuación. El tanque se dirigiría

inmediatamente a esa casilla, y en el momento en que la alcanzase, se volvería a hacer el cálculo de las casillas cercanas, teniendo en cuenta que pueden haber cambiado la zona objetivo, la estrategia y los amigos y enemigos próximos.

Después de este estudio se crearon unos "factores" que multiplicaban cada aspecto de los cinco a evaluar, de manera que si queríamos cambiar rápidamente la puntuación correspondiente a altura, desnivel o lo que sea, solo tendríamos que alterar esos factores. Esto se convirtió así en otro problema de reconocimiento de patrones, ya que teníamos unas variables que podían tomar diversos valores y teníamos que buscar los que resultasen en una mejor solución para la aplicación. Pero antes de tomar una decisión sobre la solución se analizaron las implicaciones del problema. Los casos iban a ser muy variados, ya que la influencia de la estrategia, los aliados y los enemigos iban a cambiar mucho durante el combate. Por lo tanto, resultaba muy difícil encontrar unos datos que fueran a funcionar bien en todos los casos a los que el tanque se enfrentase, y por esto se planteó que no iba a ser tan relevante escoger unos valores óptimos en esta parte del proyecto. Por otro lado, podía suponer una pérdida de tiempo que no mejorase el sistema de inteligencia artificial global.

La opción que se tomó en este primer problema de reconocimiento de patrones del proyecto fue la del enfoque estructurado: analizar de forma superficial cómo afectan estos valores a las decisiones de los tanques y ajustarlos manualmente hasta que el resultado fuera bueno. Después de algunas pruebas, los parámetros estaban bien ajustados. Los tanques valoraban la altura con bastante fuerza restándole algo de desnivel, pero a la vez trataban de alejarse de los enemigos y aliados para no colisionar.

El autómata se revisaba cada segundo de la aplicación; en esta actualización se volvían a valorar las casillas cercanas a cada tanque según las puntuaciones definidas. El autómata realiza esta tarea, de manera que el tanque solo debe tomar la casilla mejor puntuada, cada segundo de la aplicación, o cuando no tenga una casilla con máxima puntuación. Esto se puede producir al inicio del programa o cuando se produzca un evento que desencadene este aspecto, como puede ser el alcance de una casilla objetivo o la destrucción del tanque al que se estaba disparando o incluso un cambio de estrategia, producido por ejemplo por un disparo recibido que incita al tanque a huir de la zona. Este sistema resolvió el problema de la inteligencia artificial en el ámbito cercano al tanque, de manera efectiva y sin agotar recursos.

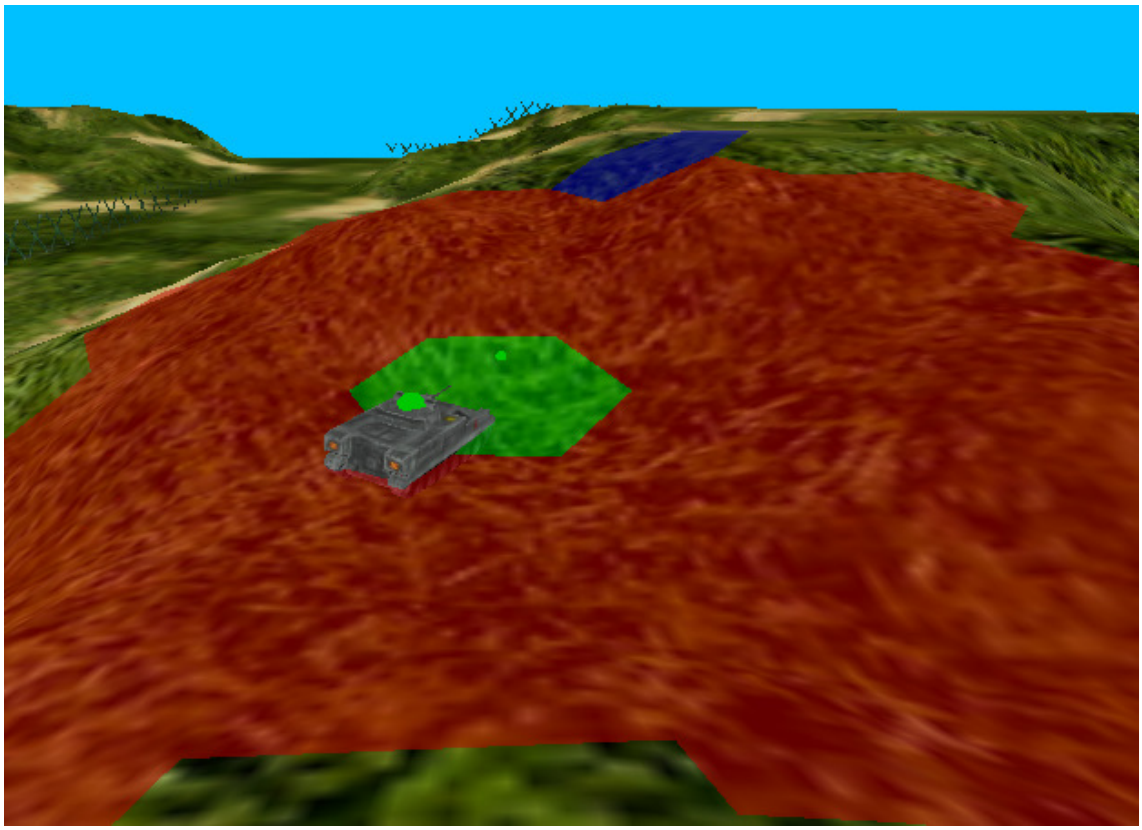


Imagen 3.15: De las casillas cercanas (rojas) el tanque escoge la de más puntuación, en este caso, la más alta (azul). La casilla en la que está situado el tanque (verde) no se puntúa

Después de que ya hubiese valoración de casillas en el simulador, se pasó a estudiar el problema del modo de movimiento de los tanques, más concretamente a lo referido a colas de movimientos; parece lógico pensar que siempre será útil e interesante poder apilar movimientos al tanque, de manera que pueda patrullar una zona, moverse en zig-zag o simplemente rastrear el terreno. Sin embargo, dado el sistema de puntuación de casillas que escogido, siempre habría una única casilla a la que se dirigiría el tanque en cada momento. Si quisiéramos que realizase movimientos complejos, habría que tratar con las zonas objetivo según la estrategia vigente, de manera que el tanque se fuera moviendo por las zonas adecuadas. Pero como ya se ha mencionado, esto sería cosa de la estrategia, que puede que sí debiera poder apilar movimientos, pero esto no se aplicó a los movimientos finales del tanque. Con esto consiguió que el tanque fuera más dinámico, reactivo e impredecible, ya que sería muy triste que por haber decidido de antemano los movimientos a realizar, el tanque se dirigiese inevitablemente a una muerte

segura o que perdiera cualquier ventaja ganada. Además este sistema de movimiento resultó más ligero y fácil de implementar.

También se abordó en esta fase el problema básico de visión del tanque; para detectar enemigos es preciso que el tanque "vea" una zona de terreno. Gracias a la función del motor grafico "IsVisible" podemos ver si 2 entidades se están viendo, lo que simularía de manera bastante buena nuestro problema. El problema de IsVisible es que no entiende de distancias, por lo que 2 tanques muy lejanos se verían sin problemas y el simulador sería menos interesante. Sin embargo se le puede aplicar manualmente el tema de la distancia, especificando que si el objeto visto está a más distancia de X metros lo ignore por completo, actúe como si no lo viera. Esto puede parecer dar ventaja a los humanos reales, ya que ellos si que ven el tanque desde lejos, pero con el comando "EntityAudoFade" se puede hacer que la distancia a la que se dibujen los tanques con respecto a la cámara sea la misma a la que ven los tanques, y con este sencillo comando se solucionó el problema.

Para apuntar, se hizo que los tanques giraran la torreta hacia el enemigo si es que “veían” un enemigo, y si no veían a ninguno, simplemente ponían la torreta alineada con el tanque. Además, si estaban apuntando a un enemigo automáticamente disparaban para acabar con él. Es lo que se llamó “ver y protegerse”, y se incluyó en los tanques de manera que permaneciese activo en cualquier instante de la simulación.

En esta fase también se planteó la forma de distinción de los tanques para identificar a los amigos y los enemigos. Aplicar un algoritmo de visión artificial sobre los tanques enemigos sería lo más adecuado, pero se disponía de los medios para hacer esto así de modo que se dejó pendiente para los ajustes finales. Por el momento, los tanques identificaban automáticamente si los otros vehículos eran amigos o enemigos y actuaban en consecuencia. Para ello, simplemente miraban un atributo de equipo en ambos tanques y comprobaban si eran iguales o diferentes. El problema de la visión artificial sería tratado aparte. La razón era que acoplarlo al simulador resultaba muy lento por lo medios de software con lo que fue modelado, pero ya se verá cómo se estudió este problema con más detalle.

Aquí terminó la creación de la inteligencia artificial local de la aplicación. Quitando el hecho de que los tanques se dirigían hacia zonas aleatorias del mapa en algunos momentos determinados, se veía bastante bien

como valoraban el terreno, se encontraban en algún momento y se disparaban hasta la destrucción. Los resultados de esta fase de la implementación fueron bastante buenos, habiendo conseguido ya un nivel de inteligencia artificial bastante digno y que solo necesitaría algunos ajustes de la siguiente fase para resultar óptimo. Las pruebas realizadas revelaron que los tanques se seguían comportando bastante bien, aunque se veían algunos fallos. Por ejemplo, los tanques se quedaban atascados en las fronteras o esquinas del mapa, aspecto que se solucionó al instante con un sencillo algoritmo. Un problema más grave era que los tanques solían chocarse (aunque se intentó que esto no ocurriese) y se quedaban atascados (Imagen 3.16). Además, si esto ocurría entre 2 tanques enemigos, no eran capaces de dispararse, por lo que el atasco se prolongaba hasta un tercero acababa con uno de ellos. Este fallo, que parecía más fácil de resolver de lo que en realidad fue, se trató en el siguiente apartado del proyecto.

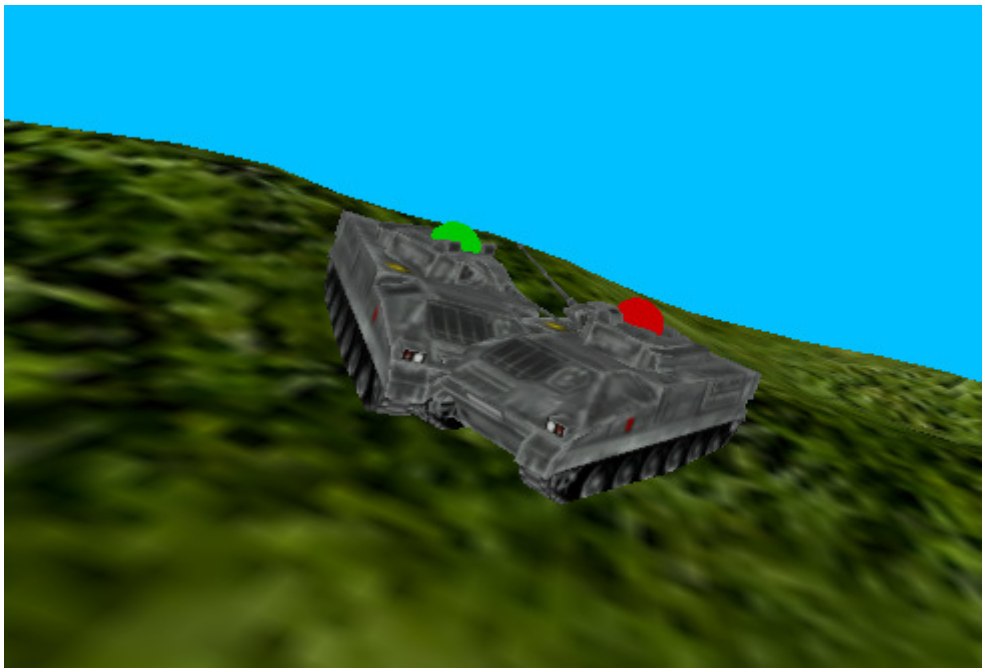


Imagen 3.16: 2 tanques se enemigos chocan y se quedan atascados. Tan pegados no pueden destruirse

Fase 4: inteligencia artificial global y mejoras

La inteligencia artificial local creada en la fase anterior ya proporcionaba a los tanques un buen sistema de movimiento por las cercanías de su zona actual, pero los tanques debían dirigirse a algún sitio de forma global en el mapa, y esta decisión había de ser inteligente y coherente en este simulador. En la fase 3 esta zona objetivo se determinaba aleatoriamente entre todas las casillas del mapa, siendo esta una de las peores soluciones que se pueden diseñar, pero que sirvió como medida provisional para poder ir desarrollando la aplicación. Así que en este apartado se propuso aplicar alguna técnica de inteligencia artificial para mejorar esta elección, poniendo así el broche final a este proyecto. También se incluirían en este apartado los arreglos finales para mejorar la experiencia del simulador, que no son muchos porque el objetivo de este proyecto no es conseguir una simulación buena en todos los aspectos, sino la creación de una buena inteligencia artificial con los medios de que se dispone.

En el análisis de esta fase se determinó la inteligencia artificial global (para todo el mapa) se generase a partir de la estrategia concreta que siguiese el tanque en cada momento, de los datos del mapa y de los aliados y enemigos localizados. El resultado devuelto por este sistema de inteligencia sería una casilla concreta del mapa, que serviría de guía o meta para los movimientos del tanque correspondiente

El diseño para conseguir acoplar estos requisitos al programa se resolvió de la siguiente forma. Para las estrategias bastaría con añadir un atributo a la estructura tanque que reflejase la estrategia que tenía cada tanque en cualquier momento de la partida. En cuanto a la inteligencia artificial, la idea era alterarla de alguna manera para que fuera muy simple y se pudiesen organizar batallas de muchos tanques sin que eso afecte al rendimiento del juego. Además había que completar el algoritmo inicial para que respondiera a órdenes más específicas y se pudiera interactuar con él desde fuera del juego o desde otro tanque aliado.

¿Cómo crear inteligencia artificial simple? El problema principal del algoritmo complejo es que periódicamente y cada poco tiempo cada tanque rastreaba el terreno en busca del mejor camino para llegar a su objetivo. Esto lo hacía cada tanque porque el camino a elegir a cada destino depende directamente de la posición en el mapa. Sin embargo, para crear un algoritmo rápido para muchos tanques se podría simplemente realizar un escaneado

rápido y general del terreno periódicamente (por ejemplo, cada 10 segundos) en el que se guardara en una tabla los valores de las casillas según los enemigos y aliados localizados en el mapa (al fin y al cabo, los valores del terreno no cambian, son permanentes). Este sistema además se combinó con el algoritmo anterior para crear un programa muy efectivo en el campo de batalla. Así que en lo que al análisis se refiere, se optó por añadir una función que calculase la zona objetivo del tanque cada 10 segundos, del mismo modo en que el algoritmo de "Ver" de los tanques se ejecuta más o menos cada segundo.

Tras haber repasado y confirmado el análisis y diseño, comenzó el periodo de implementación de esta fase final de la aplicación. Para determinar la zona hacia la que se dirigía el tanque, se tomó una única casilla; la zona consistiría en las casillas cercanas a esa misma. Esto daría ventajas a la hora de, por ejemplo, dirigirse a la zona del enemigo, si se conoce la casilla en la que se encuentra en un momento dado. Lo mismo pasaría con un aliado si quisiéramos dirigirnos a su zona para apoyarle. Por lo tanto, el atributo del tanque que marcó la "zona objetivo" fue un número de casilla. Como con la casilla objetivo, en caso de ser cero significaría que no hay una zona objetivo distinta a la que se quiera ir.

Para poder controlar el rendimiento de la aplicación conforme se complicaba la IA se creó un medidor de FPS que indica si se sobrecargan los recursos del ordenador en cada instante del juego. Además se llevó a cabo un encapsulamiento de la inteligencia artificial ya creada para poder controlar y aplicar de manera más efectiva los algoritmos diseñados. El proceso de crear un medidor de FPS o Frames Per Second (número de pantallas que se crean en un instante) solo consistió en contar las iteraciones del bucle en un lapso de tiempo que controlamos con la función de tiempo que accede a la hora del ordenador. Con la precisión adecuada no supuso ningún misterio crear este tipo de función auxiliar.

Una vez funcionando, había que guardar de alguna manera la información que queríamos evaluar para poder valorar la mejor zona objetivo de todo el mapa. Así, cada tanque debería tener una lista de todas las casillas del mapa, con los valores de altura y desnivel fijos (pues estos no cambian) y que se fuera actualizando con las posiciones de los aliados y los enemigos localizados, que sí cambian a lo largo del tiempo. Esto se guardó

en una gran tabla o matriz, de manera que para calcular las mejores casillas de cada enemigo solo había que recorrer las columnas de la matriz para quedarnos con sus valores máximos.

A la hora de crear esta matriz con valores del terreno fijos, se advirtió que la altura ya ha sido recogida en la matriz de hexágonos en las fases de desarrollo anteriores, en el cuarto valor de dicha matriz. Como queríamos crear un algoritmo simple, no recogimos nada más. El tema de los enemigos se afrontó, como se ha dicho, cada cierto tiempo periódico. En cada uno de esos momentos, se pregunta qué ve cada tanque para localizar a los enemigos. En cuanto un tanque enemigo sea visto, se anota su posición en el mapa con una puntuación. También se anotan las posiciones de todos los tanques del equipo. El último problema era el asunto de la distancia a cada tanque del equipo de cada casilla. Si no teníamos en cuenta las distancias, lo normal sería que todos los tanques fueran al mismo punto, cosa que no nos interesaba. Pero, ¿como abordar el problema de la distancia a cada casilla sin pasar por cada casilla? Una posible solución era usar zonas grandes, quizá también hexagonales, y considerar a la misma distancia todas las casillas de una misma zona. Pero esto entrañaba problemas, ya que sería una nueva distribución de casillas compleja y podría requerir muchos recursos y tiempo de desarrollo e implementación, aparte de otros problemas. Además, el tamaño de las zonas sería algo importante sobre lo que pensar, no estando claro para nada el tamaño más adecuado. Así que la solución más conveniente era realizar un único rastreo de las casillas, calculando la distancia a todos los tanques en cada casilla. Si tenemos en cuenta que este rastreo se hará una vez cada varios segundos, resulta obvio que esto no afectará de manera excesiva a la velocidad de ejecución del programa.

Esta solución implicaba la inclusión de una nueva tabla de distancia, con tantas columnas como tanques haya en el juego y tantas filas como casillas contenga el terreno. Lo hizo fue aplicar estos cálculos a la ya creada tabla de tanques y casillas. Sobre esa misma tabla, cada X segundos aplicaríamos al terreno (fijo) la distancia calculada, los enemigos vistos y los aliados. El resultado mayor de la columna será la casilla objetivo para los siguientes X segundos del tanque. Se estudio el hacer el cálculo antes de tiempo si se producen eventos importantes como la destrucción de algún tanque, amigo o enemigo, pero se prescindió de esto, ya que haría más pesadas las operaciones cuando lo que queremos es un algoritmo de cálculo de de objetivos cooperativo lo más ligero posible.

Programando esto último surgió un tema importante en cuanto a los resultados de eficiencia y velocidad del programa; resultaba que el algoritmo de selección de casillas no era lo único que ralentizaba el programa, sino que también influía mucho el algoritmo de visión diseñado. Al fin y al cabo, en cada iteración sin excepción comparaba la visibilidad mutua de sus partes con la del resto de tanques, haciendo pesados cálculos que generaban lentitud. Así que se volvió a plantear el problema de la visión. La primera idea valorada era la de retomar la idea de los conos de visión, solución parecida a la que algunos motores gráfico utilizan para representar las sombras geométricas o "Stencil". Sin embargo, la innovación sería el hacer "chocar" el cono con el terreno para ver que casillas están en contacto con el cono, pues esas casillas serían las que el tanque estaría viendo. Parecía una solución bastante difícil y problemática, cuando en realidad el problema es que las consultas de visibilidad se hacían unas 60 veces por segundo (o las que se puedan hacer, según los FPS). Si se frenase el cálculo de la visión a una única vez por segundo (aproximadamente) no se obtendría tanta lentitud, y esto afectaba a la efectividad de los tanques (no eran tan rápidos). Esta implementación funcionó muy bien, tal y como se esperaba. Además, permitía tener activada la visión/protección siempre, sin que eso afectase al rendimiento. Esto significa que los tanques siempre iban a ver lo que tenían delante y a protegerse consecuentemente.

Aunque se hubo aligerado el algoritmo de ver, el de calcular las celdas seguía siendo bastante pesado, ya que el programa solo funciona a plena potencia cuando había cinco o menos tanques ejecutándolo y viendo (pues el algoritmo de ver todavía influía en el rendimiento global). Así que el algoritmo de cálculo de objetivos por equipo cada X segundos seguía siendo una alternativa razonable y por lo tanto se procedió a implementarlo.

El comportamiento del motor gráfico en este punto del proyecto resultaba extraño. Cuando se forzaba el juego los FPS descendían de 63 (rendimiento máximo) a 32. Y lo más raro era que el juego se mantenía en 32 aunque se sometiese a mucha carga de cálculos. Esto me llevó a pensar que quizá se podrían acelerar los tanques y el resto de aspectos del juego para que a 30 FPS funcionasen de manera correcta. Por supuesto, cuando los tanques fueran muriendo el juego funcionaría a 60 FPS e iría demasiado rápido para el jugador / observador, por lo que se redujo el funcionamiento del juego a los FPS que queríamos (30) usando la función Pause (Delay), que detiene la ejecución del programa. Se consiguió así que el juego

funcionase a la velocidad adecuada durante todo el tiempo, pudiendo ejecutar los cálculos complejos más conveniente (dentro de unos límites), resolviendo así gran parte de los problemas surgidos.

En cuanto a las zonas objetivo o la manera de moverse por el terreno, se trató esto como un problema de reconocimiento de patrones; aunque ya teníamos un cálculo con respecto a la distancia y la altura, se incluyó aquí la posición de amigos y enemigos y la estrategia vigente en cada vehículo. Así que para decidir qué técnica de inteligencia artificial era más conveniente en este otro problema se estudiaron sus características. Como programa en tiempo real en el que no se aceptaban paradas, los ya realizados cálculos de tratamiento de los gráficos y del resto de entidades de la aplicación tampoco permitían un sistema que requiriese mucha carga de procesamiento, y había que adaptarse a los medios de que se disponibles. Por estas razones se descartaron las técnicas de redes neuronales, lógica difusa y algoritmos genéticos, para este otro problema a resolver, y se enfrentó de manera estructurada, ajustando manualmente qué harían los tanques en cada caso. Además, no estaba muy claro que con los valores óptimos se fuera a comportar la aplicación perfectamente en cada caso, así que para evitar problemas de programación y de recursos en tiempo de

A la hora de incluir la posición de amigos y enemigos en las valoraciones del mapa, se vio que el problema se podía resolver de manera muy simple: puntuando únicamente las casillas en las que se encontraran en un momento dado los enemigos. Esto era posible ya que luego el algoritmo de movimiento cercano buscaría las mejores rutas y la mejor casilla a la que dirigirse en la zona que le hemos indicado. Este aspecto también se podía aplicar a la posición de los aliados, al menos en caso de querer acercarse o alejarse de su posición.

La pregunta pertinente fue cómo puntuar las casillas con aliados o enemigos. Y la respuesta es obtenida era dependía de la estrategia que seguía el tanque en ese momento concreto. En ese momento se definieron las estrategias a implementar; aunque a priori se idearon muchas y muy variadas, se decidió implantar unas pocas para este proyecto. Al fin y al cabo, las estrategias son algo que se puede añadir siempre que se quiera para completar/mejorar la inteligencia artificial. Nos decidimos por algunas

estrategias de las más fáciles de implementar pero que resultaban vistosas y efectivas en el combate.

ESTRATEGIAS:

- **Huida:** En primer lugar se generó la estrategia de huir; cuando al tanque le quede poca vida tratará de alejarse de sus enemigos y también de sus aliados, ya que seguramente atraerán o buscarán nuevos enemigos.

- **Despliegue:** Cuando no haya enemigos a la vista, será interesante la estrategia de posicionarse en el terreno; se buscarán los puntos más elevados del mismo a una distancia razonable. Los puntos altos, aunque puedan representar peor posición de disparo permiten ver mayor cantidad de terreno para localizar enemigos.

- **Ataque:** Por último tendríamos la de atacar, en la que se intentará mantenerse junto a los aliados mientras se avanza hacia posiciones enemigas. Sería algo así como un ataque en grupo simultáneo (sólo si los demás también llevan esa estrategia).

Se concretó que con estas estrategias sería suficiente para crear unas buenas simulaciones de batalla. Baste decir que en estas estrategias no solo se utilizan como parámetros las puntuaciones de la tabla de valores, sino también las de moverse en la zona concreta en la que se encuentre el vehículo, de manera que estos valores también son alterados según la estrategia vigente.

Es estudió en este punto cuándo realizar los cambios de estrategia. Tiene sentido que cuando se detecta a un nuevo enemigo o cuando se destruye a un aliado o enemigo o cuando se tiene poca vida (huir) se cambie la estrategia, aunque no afecte al movimiento del tanque hasta unos segundos después. Esta detección se separó para los dos equipos o facciones de tanques, ya que es posible que un tanque vea a un enemigo “por la espalda” sin que se de cuenta. En ese caso, el equipo que no ha visto nada no cambia de estrategia porque sigue en la misma situación. Como el algoritmo

de ver funcionaba cada segundo (y no cada frame) no se generarían excesivos cambios de estrategia, y si era el caso, bastaría con aumentar la vida de los tanques para que no hubiera tantos cambios. Huelga decir que, por ejemplo, una vez llegado a l estado de huir no se cambiaría de estrategia, al menos hasta no detectar enemigos, cosa que también favorecería los pocos cambios de estrategia.

El resultado final del algoritmo de inteligencia de esta fase fue un código rápido y efectivo que cumplía con los requisitos establecidos al inicio de este ciclo. En todo momento los tanques conocen la posición del resto de vehículos de su equipo; se correspondería con que se la fueran comunicando por radio o que tuvieran algún sistema de radar. Cuando cada segundo los tanques "ven", si detectan algún enemigo se lo comunican al resto de los tanques, poniendo su posición en una tabla común que todos consultan. Cuando la tabla se actualiza, ya sea por una destrucción de aliados o enemigos o por una detección de algún rival, los tanques determinan su estrategia basándose en las localizaciones que figuran en la tabla.

Este problema, que también es de reconocimiento de patrones, porque hay que estudiar unos valores para realizar una toma de decisión, se resolvió de nuevo de manera estructurada. Otra vez es un entorno demasiado dinámico y rápido como para poder aplicar técnicas más potentes como redes neuronales o lógica difusa y obtener buenos resultados sin afectar al funcionamiento. Además, este problema era más fácil de representar y resolver. La estrategia de huir solo se elegiría si se tiene menos de un cuarto de la vida o blindaje del tanque y no se cambiaría a ninguna otra estrategia posteriormente. La estrategia de posicionamiento o despliegue solamente se ejecutaría si no se detectan en ese momento enemigos, mientras que la estrategia de atacar se activaría en caso contrario. Visto así no parecía un problema de reconocimiento de patrones, pero esta fase del proyecto se diseñó para que se pudieran incluir multitud de nuevas estrategias. A la hora de elegir entre otras estrategias es posible que no fuera tan fácil distinguir los casos para cada una de ellas, con lo que el problema se convertiría en un auténtico problema de reconocimiento de patrones.

Con las estrategias ya establecidas y valoradas, el programa calcula la zona objetivo más adecuada a la que dirigirse. Para ello, cada diez segundos aproximadamente, cada tanque coge su tabla de valores general del terreno, en la que se indican las alturas del mapa, y le aplica una puntuación que puede ser negativa o positiva de la distancia a cada tanque que tengo

localizado en el terreno. Esto depende de la estrategia, ya que por ejemplo en la estrategia de despliegue se resta puntuación por las casillas con tanques aliados para no coincidir en las mismas zonas, pero en una hipotética estrategia de defensa, los aliados puntuarían positivamente para incitar a la defensa en grupo. También se resta un factor de distancia a la casilla actual del tanque, para dirigirse a lugares no muy lejanos. Esta puntuación sin embargo no es tan alta como las anteriores, porque por ejemplo tiene prioridad durante la estrategia de ataque avanzar hacia el enemigo por muy lejos que esté. Tras acabar las puntuaciones, cada tanque calcula la puntuación máxima de la tabla, cuya casilla relacionada es la que establece como zona objetivo. A partir de ese instante, el tanque se dirigirá hacia esa zona, respetando siempre los caminos escogidos por la inteligencia artificial local dirigida por el autómata celular.

Imagen 3.17: Captura de pantalla de la aplicación finalizada

Así terminó el proceso de dotar de inteligencia a los tanques del simulador. Las soluciones implementadas consiguieron que los tanques se comportasen de manera bastante “humana”; no actúan perfectamente y ni siquiera inmediatamente, pero el efecto que producen es que los tanques tardan en comunicarse y en coordinarse para reaccionar a los cambios producidos. Además los cambios de estrategia se ejecutan a frecuencias bastante parecidas a los que adoptaría una persona humana, aunque no sean tan inteligentes y astutos.

La fase de pruebas que se realizó en este momento para testar el código y que consistió en comprobar que los tanque actuaban de la manera diseñada y fijada, reveló que era muy frecuente que los tanques se chocasen en su rutinas y no pudieran separarse. Al chocar se quedaban bloqueados y ni siquiera se podían disparar si eran enemigos. El problema era que aunque sus casillas objetivo fueran distintas, podían coincidir en su camino hacia ellas, lo que hacía bastante difícil controlar el que no se chocaran. Una solución era plantearse alguna reacción ante el choque, esto es, que por ejemplo cuando se chocasen dieran marcha atrás unos metros para evitarse en su camino. Esto podría resolver bien el problema, ya que si son enemigos es posible que se ataquen en esa marcha atrás temporal y acaben destruyéndose. Otra opción evaluada era tratar de evitar que los tanques se acercasen a casillas cercanas a otros tanques, prohibirlas. Esto es lo que se decidió hacer, y se implementó en cada ciclo midiendo la distancia de cada casilla al resto de tanques, como se hace al "localizar tanques". Luego la casilla se marcó como prohibida y los tanques la intentaban evitar. Además se redujeron las casillas que se evaluaban alrededor del tanque para que no se metiera en casillas prohibidas de camino a casillas permitidas. Se guardó además el número del tanque para saber cuál es el que está más cerca de la casilla y prohibirla a todos excepto a ese mismo tanque, cosa que no tendría sentido. Una nueva secuencia de pruebas reveló que los tanques ya casi no colisionaban, cosa que solo sucedía en ocasiones puntuales, y que también simula otro aspecto de las batallas de tanques. Al fin y al cabo, solía suceder que los tanques se chocaran y quedaran en corta distancia, sobre todo en terrenos pequeños, estrechos o muy obstaculizados, como pueblos o ciudades.

Con este último problema resuelto finalizó el proceso de implementación de la fase final (Imagen 3.17). El resultado fue un simulador en el que los tanque entran por sí solo en batallas, acuden a las zonas más altas del terreno, dispersándose si no ven enemigos. En cuanto ven un enemigo, se van dirigiendo a esas zonas y allí buscan los mejores lugares para atacar. Cuando tienen poca vida, abandonan la lucha.

Fases paralelas: algoritmo genético y reconocedor de tanques

Aparte de la aplicación principal del proyecto, que consiste en un simulador de batallas de tanques con inteligencia artificial funcional, se aplicaron otras técnicas a algunas partes del proyecto para estudiar sus resultados en entornos controlados. No se acoplaron a la aplicación principal porque requerían de recursos de procesamiento que no se pueden conseguir sin ralentizar el simulador y porque su realización resultaba extremadamente complicada para con el programa creado. Trabajar en un entorno más simple, optimizado y con menos detalles era necesario para poder realizar estos proyectos.

En la primera de las subfases se aplicó un algoritmo genético para determinar los mejores valores de algunos atributos propios de los tanques. Se jugó con la velocidad de movimiento, de disparo, el alcance de los proyectiles, el daño de los mismos y la vida o blindaje del tanque. Había una cierta cantidad de puntos para distribuir entre estas variables, de modo que si se invertían en una no se podían poner en las demás. Esto aseguraba que los combates fueran "justos", ya que todos los modelos de tanque partían con el mismo número de puntos iniciales.

Imagen 3.18: Captura de pantalla del entorno controlado en el que se aplicó el algoritmo genético

El entorno en que se realizaron estas pruebas era una batalla en dos dimensiones de “uno contra uno”. Esto aseguraba que no influyesen en los combates factores que no tenían que ver con las variables de los tanques, como una repentina superioridad numérica o un problema con los desniveles del terreno.

Se partió de una configuración aleatoria de las variables genéticas y se realizaron multitud de combates; se probaba cada configuración contra muchas combinaciones aleatorias del tanque enemigo, y se puntuaba el resultado global. Si había mejorado la puntuación de la configuración anterior, ésta última quedaba como lo buena, y si no, se volvía a los valores inmediatamente anteriores. Después se “mutaba” la secuencia introduciendo algún cambio no muy significativo, esto es, quitando unos pocos puntos a una variable y poniéndoselos a las demás (o a otra concreta).

El resultado fue una configuración de las variables genéticas que puestas en un tanque ganaba fácilmente la mayoría de partidas contra tanques con valores genéticos aleatorios. Más concretamente, se valoraba mucho el alcance de los disparos, ya que pudiendo disparar antes que el enemigo se conseguía una clara ventaja. El parámetro más irrelevante y por lo tanto menos puntuado era la velocidad de movimiento, ya que no se conseguían evitar los disparos por velocidad y no servía para mucho más. Estos resultados se corresponden de alguna manera con la realidad de los carros blindados y los tanques. Por ejemplo, en la Segunda Guerra Mundial, los ingenieros alemanes crearon una nueva y potente versión del Panzer IV a la que incorporaron un cañón antiaéreo 88: el tanque Tiger. El tanque era desde luego más lento que otros más ligeros y tenía menor velocidad de disparo, pero su largo alcance de tiro, de decenas de metros más amplio que el los tanques americanos Sherman, hizo que se convirtiera en el vencedor de muchas contiendas y batallas en el norte de África.

Estos resultados fueron utilizados en este proyecto a la hora de determinar las características básicas de los tanques. Como se puede ver en la aplicación, el alcance del cañón permite disparar a largas distancias, y si los tanques no se disparan de más lejos es porque se redujo su campo de visión. Esta medida se tomó porque el campo de batalla es más bien pequeño (para que los tanques se destruyan rápido) y si no se matarían desde lejos sin moverse del sitio. También se estableció que los tanques fueran más o menos lentos; al fin y al cabo, como el mapa es pequeño, no hay necesidad de hacer que se desplacen muy rápido. Sin embargo, merece la pena más que disparen rápido, y esto es lo que hacen.

La otra aplicación de inteligencia artificial que se realizó paralela a este proyecto fue aplicar redes neuronales a la visión artificial de los tanques. Se realizó fuera del proyecto porque, como se vio al final de este trabajo, requería bastantes recursos de procesamiento realizar estos esfuerzos. Lo que se pretendía en concreto era determinar si el tanque era aliado o enemigo a partir de una imagen del mismo. Para esto se tomaron imágenes de los tanques sobre un fondo con bastante contraste y sin ningún elemento más, para que interfirieran con los cálculos de la red neuronal. La idea era que la red neuronal, a partir de una imagen tomada desde cualquier ángulo del tanque, pudiera decir si el tanque era de un tipo u otro. Se tomaron dos modelos de tanque muy distintos para facilitar esta decisión (Imagen 3.19); al fin y al cabo, ya iba a haber bastante dificultad como para usar tanques parecidos.

Imagen 3.19: Modelos de tanque utilizados en el algoritmo de reconocimiento de tanques con redes neuronales.

El tratamiento de las imágenes se realizó con Matlab, ya que se requería un algoritmo de extracción de bordes y el motor gráfico Blitz3D no dispone de nada parecido. La interacción con Matlab produjo problemas desde el principio, pero se logró solventar con un acceso a un ejecutable que tardaba más tiempo del debido en ejecutarse. En Matlab era muy rápido, pero al convertirlo en ejecutable experimentaba grandes retardos, así que este módulo no se acoplo al proyecto final.

Así, tras obtener una foto o imagen de un tanque, se escalaba, se obtenía el contorno y se binarizaba para obtener un borde blanco sobre un fondo negro.

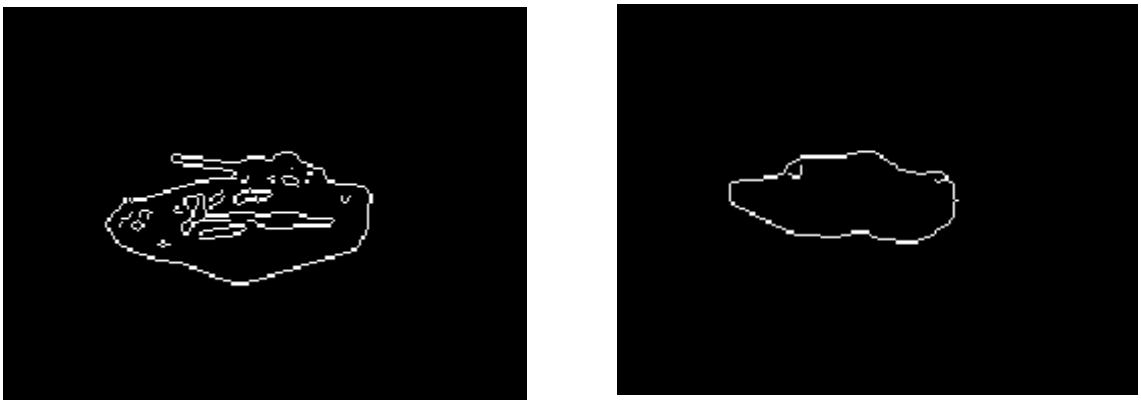


Imagen 3.20: Imágenes con las que trabaja la red neuronal del Reconocedor de Tanques

Esta información es la que se introducía en la red neuronal, que había entrenado con fotos de 32 ángulos de los tanques. Así, aunque la foto tomada fuera de un ángulo nunca visto exactamente por la red neuronal, podía hacerse una idea de cuál era el tanque que se le pedía identificar. La red funcionaba muy bien, con más del 80% de acierto en las pruebas, cuyos casos se añadieron también a la red para que estuviese todavía mejor entrenada. Se creo además un programa que permitía al usuario girar el tanque y realizar una comprobación instantánea (aunque un poco lenta por la interacción con Matlab) para ver si el programa conseguía reconocer o no el tanque. El aspecto de la aplicación se puede ver en la Imagen 4.3.

Imagen 3.21: Aspecto de la aplicación independiente Reconocedor de Tanques

Este fue el trabajo adicional que se realizó para mejorar el proyecto. Estos dos mini proyectos se pueden también ampliar en líneas futuras, o incluso fusionarlos con el proyecto principal, consiguiendo una mejora significativa del programa.

4. Líneas futuras y conclusiones

Se pueden imaginar muchas líneas futuras para este proyecto. Podría especializarse en crear un simulador de batallas de tanques, aplicado quizá a un modelo de tanque concreto y actual, para servir de formación a conductores. Podría derivar en un videojuego propiamente dicho, creando niveles o batallas personalizadas o incluyendo más armamento, unidades o vehículos.

Por otro lado, se pueden mejorar desde el apartado gráfico y de manejo de la aplicación (ahora es un poco primitivo), hasta la inteligencia artificial, mejorando las técnicas aplicadas o introduciendo otras nuevas. Esto último ya ha sido realizado en paralelo a este proyecto, como vimos en el apartado anterior. Este trabajo no ha sido introducido dentro del desarrollo del proyecto como tal, ya que en vez de aplicarse sobre la aplicación en sí se crearon otros entornos más controlados para probar estas ideas. Fueron dos trabajos de reconocimiento de patrones, uno solucionado con un algoritmo genético y otro con una red neuronal.

Pero la principal línea futura del proyecto (y la que pretendo implementar por mi cuenta de ahora en adelante) es la de hacer un juego funcional para el usuario. Es bien sabido que no hace falta que los juegos sean gran cosa en cuanto a gráficos o cantidad de elementos, sino que sean divertidos y que resulten retos para el jugador [5]. Por ejemplo, “Angry Birds” o “Plants vs. Zombies” son títulos con calidad gráfica y de procesamiento reducidas pero que gustan a la mayoría de usuarios y por eso se han hecho tan famosos. También los clásicos siguen gustando, desde juegos de la Nintendo NES como “Super Mario Bros” o “Battletoads”, de Sega Megadrive como “Sonic” o de Nintendo 64 como “Star Fox 64”, entre muchos otros. Estos juegos, con muchísimas limitaciones por la tecnología existente en los años 90, todavía siguen gustando a los jugadores, que no se desprenden de ellos. Esto permite a los creadores independientes de videojuegos tener la oportunidad de entrar en el mercado aunque sus juegos sean de menos tamaño y peor calidad que los que publican las grandes empresas del sector. Así que el proyecto evolucionará para convertirse en un videojuego simple pero entretenido, que suponga un reto para el jugador de manera que pueda disfrutarlo, aunque sea un rato, o al menos esa es la idea actual sobre el futuro de la aplicación.

En conclusión, hemos comprobado que las técnicas de inteligencia artificial sobre reconocimiento de patrones aprendidas durante la carrera han funcionado muy en un entorno creado artificialmente pero que simula las condiciones reales en un entorno concreto como son las batallas de tanques. Los tanques se comportan adecuadamente, con inteligencia y coherencia pero sin abusar de su capacidad de cálculo. En algunas ocasiones los tanques parecen liarse un poco y moverse inteligentemente, pero esto podría pasarle a un humano nervioso por el desenlace de la batalla; además, conseguir una inteligencia artificial perfecta es muy difícil hasta en proyectos grandes y muy importantes, y como ya he dicho, podría hasta quedar artificial e incluso frustrante para el usuario. Lo importante es que la inteligencia artificial del juego ha quedado bastante bien y que los tanques se comportan de una forma muy efectiva y coherente.

Los 3 métodos utilizados, reconocimiento estructurado, algoritmos genéticos y redes neuronales, dieron buenos resultados en los problemas en que fueron aplicados. La ayuda del autómata celular fue muy interesante e importante, y simplificó la manera de diseñar la aplicación haciéndola más eficiente. Sin estas técnicas de inteligencia artificial, me habría sido muy complicado realizar un trabajo parecido en el que los objetos y aplicaciones se comportasen tan bien.

Además, parece que las elecciones de qué método usar en cada caso específico fueron bastante correctas, ya que aunque otras técnicas pudieran haber funcionado aún mejor, las aplicadas resolvieron los problemas y cumplieron los requisitos de procesamiento establecidos. El proyecto se adaptó a lo requerido en un principio y la aplicación final es vistosa, funciona bien y concede una buena experiencia a los usuarios externos que la han visto funcionar.

La herramienta principal utilizada, el motor gráfico Blitz3D, fue un éxito, ya que permitió realizar toda la aplicación sin suponer ningún problema importante y facilitó las cosas en mucho casos gracias a su buena documentación y a sus ejemplos interactivos y ejecutables. En el "Reconocedor de Tanques" realizado con una red neuronal, su integración con Matlab falló, pero no fue culpa de Blitz3D, ya que su lanzador de ejecutables funcionaba bien, sino más bien de la herramienta de exportación de Matlab, que genera ejecutables muy lentos al no usar su motor interno. Sin embargo, esta elección fue mía y por lo tanto es un error.

La experiencia del proyecto ha sido muy satisfactoria. Aunque había momentos en los que la creación de algoritmos era complicada y extensa, el resultado era muy motivante. Ver funcionar las cosas de manera gráfica e incluso poder jugar contra los entes inteligentes que uno mismo ha creado es una buena experiencia. Espero seguir trabajando con este motor y para la realización de juegos o simuladores y espero aplicar las mismas técnicas de inteligencia artificial y reconocimiento de patrones que he utilizado e incluso otras, como lógica difusa, que valoré pero no llegué a implementar.

5. Bibliografía

- [1] REDES NEURONALES ARTIFICIALES. FUNDAMENTOS, MODELOS Y APLICACIONES. Hilera González, José Ramón. Ed Microinformática

- [2] LÓGICA BORROSA. Maillard, Chantal. Ed Miguel Gomez

- [3] ALGORITMOS GENÉTICOS GENERALIZADOS: VARIACIONES SOBRE UN TEMA. Iglesias Otero, María Teresa. Ed Universidad de Coruña

- [4] PATTERN CLASSIFICATION. R.O. Duda, P.E. Hart, D.G. Stork. Ed. Wiley

- [5] 3D GAME PROGRAMMING ALL IN ONE. Kenneth C. Finney. Ed André Lamothe, CEO, Xtreme Games LLC

- [6] BLITZ3D PROGRAMMING MANUAL. Blitz Research

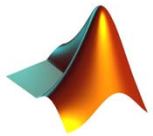
APLICACIÓN DE TÉCNICAS DE RECONOCIMIENTO DE PATRONES EN LA SIMULACIÓN DE BATALLAS DE TANQUES COOPERATIVAS

Iñaki Ugarte Gil

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Introducción

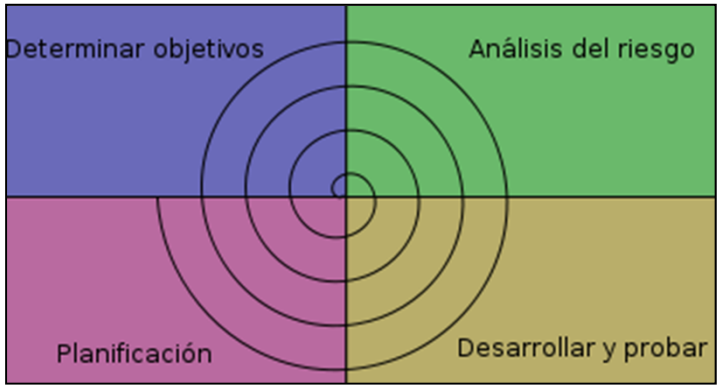
- Realizar un videojuego / simulador
- Herramientas:
  - Blitz3D → Motor gráfico
  - Matlab → Herramienta de cálculo



- Objetivo del proyecto: simulador de batallas de tanques cooperativas

Análisis y Diseño

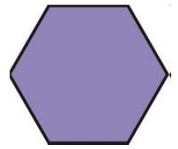
- Paradigma de desarrollo estructurado
  - Herramienta de programación:
    - Motor gráfico Blitz3D → Muchas ventajas
- Paradigma de desarrollo en espiral
  - Seguridad y eficacia
  - En cada fase de desarrollo: análisis, diseño, implementación y pruebas



Análisis y Diseño

- Estructuras diseñadas:
  - Tanque
  - Proyectil
  - Explosión
  - Malla hexágono



- Sin hilos, procesos ni herencia

Análisis y Diseño

- División del proyecto en 4 fases:
  - Fase 1: *Tanque movible sobre el terreno*
  - Fase 2: *Múltiples tanques y disparos*
  - Fase 3: *Inteligencia artificial local*
  - Fase 4: *Inteligencia artificial global y mejoras*

- 2 fases de desarrollo paralelas:
  - Algoritmo genético
  - Reconocedor de tanques

Implementación

- Fase 1: *Tanque movible sobre el terreno*
  - Carga y ajuste de los modelos
  - Interacción con el tanque
  - Ajuste del vehículo con el terreno

Implementación

- Fase 2: *Múltiples tanques y disparos*
  - Introducción de varios tanques
  - Sistema de disparos
  - Sistema de explosiones

Implementación

- Fase 3: *Inteligencia artificial local*
  - Autómata celular
  - Cálculo de mejores casillas
  - Ver y apuntar

Implementación

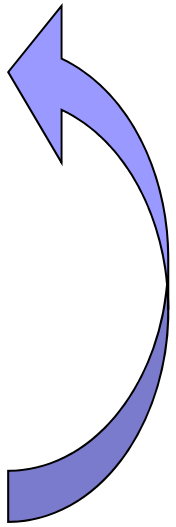
- Fase 4: *Inteligencia artificial global y mejoras*
  - Situación de aliados y enemigos
  - Estrategias
  - Cálculo de la mejor zona objetivo

Algoritmo genético

ALGORITMO GENÉTICO

- Entrenamiento:
  - Inicio con valores aleatorios

  - Series de 15 partidas
  - Mutación
  - Si el resultado es mejor
    - Guardar mutación

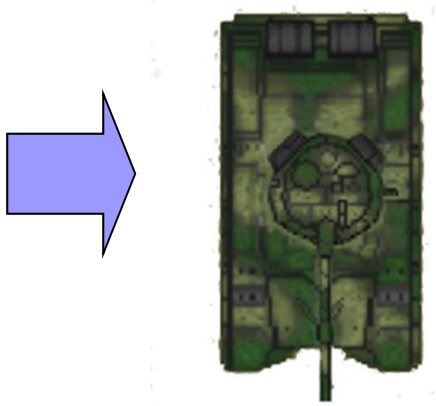


- 1500 partidas

PATRÓN FINAL

- 5 variables genéticas:
  - Velocidad → 0.00
  - Alcance → 1.00
  - Cadencia de tiro → 0.15
  - Potencia de disparo → 0.65
  - Puntos de vida → 0.70

  2.50



- Acoplado a la aplicación

Implementación

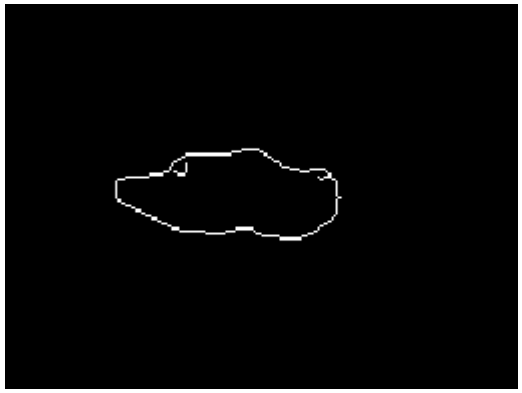
- Reconocedor de tanques
  - Preparación del entorno
  - Adaptación de las imágenes
    - Blanco y negro
    - Silueta
    - Binarización
  - Diseño y construcción de la red neuronal
    - 30.000 entradas
    - 2 salidas

Implementación

- Reconocedor de tanques

Implementación

- Reconocedor de tanques



Conclusiones

- Las técnicas de inteligencia artificial aplicadas funcionan bien
- Los tanques se comportan de manera inteligente y coherente
- Los tanques actúan en grupo de manera eficiente

Simulación final